TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Niko Siltala

Formal Digital Description of Production Equipment Modules for supporting System Design and Deployment

Julkaisu 1402 • Publication 1402

Tampere 2016

Tampereen teknillinen yliopisto. Julkaisu 1402
Tampere University of Technology. Publication 1402

Niko Siltala

Formal Digital Description of Production Equipment Modules for supporting System Design and Deployment

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Festia Building, Auditorium Pieni Sali 1, at Tampere University of Technology, on the 12th of July 2016, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2016

ISBN 978-952-15-3779-0 (printed)
ISBN 978-952-15-3783-7 (PDF)
ISSN 1459-2045

Abstract

The requirements for production systems are moving towards higher flexibility, adaptability and reactivity. Increasing volatility in global and local economies, shorter product life cycles and the ever-increasing number of product variants arising from product customization have led to a demand for production systems which can respond more rapidly to these changing requirements. Therefore, whenever a new product, or product variant, enters production, the production system designer must be able to create an easily-reconfigurable production system which not only meets the User Requirements (UR) but is quick and cost-efficient to set up. Modern production systems must be able to integrate new product variants with minimum effort. In the event of a product changeover or an unforeseen incident, such as the mechanical failure of a production resource, it must be possible to reconfigure the production system smoothly and seamlessly by adding, removing or altering the resources. Ideally, auto-configuration should obviate the need to manually re-programme the system once it has been reconfigured.

The cornerstone of any solution to the above-mentioned challenges is the concept of being able to create formalised, comprehensive descriptions of all production resources. Providing universally-recognised digital representations of all the multifarious resources used in a production system would enable a standardised exchange of information between the different actors involved in building a new production system. Such freely available and machine-readable information could also be utilised by the wide variety of software tools that come into play during the different life cycle phases of a production system, thus considerably extending its useful life. These digital descriptions would also offer a multi-faceted foundation for the reconfiguration of production systems. The production paradigms presented here would support state-of-the-art production systems, such as Reconfigurable Manufacturing Systems (RMSs), Holonic Manufacturing Systems (HMSs) and Evolvable Production Systems (EPSs).

The methodological framework for this research is Design Research Methodology (DRM) supported with Systems Engineering, Action Research, and case-based research. The first two were used to develop the concept and data models for the resource descriptions, through a process of iterative development. The case-based research was used for verification, through the modelling and analysis of two separate production systems used in this research. The concept, on which this thesis is based, is itself based on the triplicity of production system design, i.e. Product, Process and Resource. The processes, are implemented through the capabilities of the resources, which are thus directly linked to the product requirements. The driving force behind this new approach to production system design is its strong emphasis on making production systems that can be reconfigured easily. Successful system reconfiguration can only be achieved, however, if all the required production resources can be quickly and easily compared to all the available production resources in one unified, and universally accepted form. These descriptions must not only be able to capture all of a production system's capabilities, but must also include information about its interfaces, kinematics, technical properties and its control and communication abilities.

The answer to this lies in the Emplacement Concept, which is described and developed in this thesis. The Emplacement Concept proposes the creation of a multi-layered Generic

Model containing information about production resources in three different layers. These are the Abstract Module Description (AMD), the Module Description (MD), and the Module Instance Description (MID). Each of these layers has unique characteristics which can be utilised in the different phases of designing, commissioning and reconfiguring a production system. The AMD is the most abstract (general) descriptive layer and can be used for initial system design iterations. It ensures that the proposed resources for the production system are exchangeable and interchangeable, and thus guides the selection of production resources and the implementation (or reconfiguration) of a production system. The MD is the next level down, and provides a more detailed description of the type of production resource, providing 'finer granularity' for the descriptions. The MID provides the finest level of granularity and contains invaluable information about the individual instances of a particular production resource. This research involves two practical implementations of the Generic Model. These are used to model and digitally represent all the production resources used in the two use-case environments. All the modules in the production systems (25 in all) were modelled and described with the data models developed here. In fact, we were able to freeze the data models after the first case study, as they didn't need any major changes in order to model the production resources of the second use-case environment. This demonstrates the general applicability of the proposed approach for modelling modular production resources.

The advantages of being able to describe production resources in a unified digital form are many and varied. For example, production systems which are described in this way are much more agile. They can react faster to changes in demand and can be reconfigured easily and quickly. The resource descriptions also improve the sustainability of production systems because they provide detailed information about the exact capabilities and characteristics of all the available resources. This means that production system designers are better placed to utilise ready-made modules, (design by re-use). Being able to use readily available production modules means that the Time to Market and Time to Volume are improved, as new production systems can be built or reconfigured using tested and fully operational modules, which can easily be integrated into an already operational production system. Finally, the resource descriptions are an essential source of information for auto-configuration tools, allowing automated, or semi-automated production system design. However, harvesting the full benefits of all these outcomes requires that the tools used to create new production systems can understand and utilise the modular descriptions proposed by this concept. This, in turn, presupposes that the all the formalised descriptions of the production modules provided here will be made publicly available, and will form the basis for an ever-expanding library of such descriptions.

Preface

This has been a long journey, one might say too long, and contains a few complete changes of course. I must admit that I have sometimes made the 'mountain' grow even higher than it need have been. The various stops and starts have not helped either, but now it is done.

This study was carried out at the (then) Department of Production Engineering (TTE) and the Department of Mechanical Engineering and Industrial Systems (MEI) at Tampere University of Technology (TUT). First of all, I want to thank all my colleagues, both past and present, for providing a wonderful atmosphere to work in, including the moments in Tamppi -sports hall and the numerous discussions we have had on, around and about the topic, and science and technology in general.

I want to thank all my colleagues from the EUPASS project and the TUT microFactory team (Riku Heikkilä, Timo Prusi and Asser Vuola) for providing me with two platforms full of reconfigurable modules for my case studies. I also want to thank them for the many valuable and fruitful discussions we have had together, and for their friendship. The same goes for Dr. Jani Jokinen and Andreas Hofmann, whose friendship and support has been invaluable.

Next, I want to express my sincere gratitude to Dr. Christoph Hanisch and Prof. Tero Juuti, for their highly valued support and their profound insights which have often given me a much-needed 'positive push' during some of the more difficult moments on this journey, and Dr. Eeva Järvenpää, who has sparred with me during a number of fundamental discussions about the topic of this thesis.

I want to thank both of my supervisors: Prof. Reijo Tuokko for offering me the opportunity to work at TUT on so many very interesting projects and for finally guiding me towards this thesis topic and supervising the beginning of this journey; and Prof. Minna Lanz, for being so determined in supervising my thesis to its conclusion. I am very grateful to my external evaluators, Prof. Marco Taisch from Politecnico di Milano, Italy and Dr. Tommi Karhela from VTT, Finland for their comments and suggestions. I would like to thank Adrian Benfield for proofreading the text, and Syed for giving a hand in getting the final tweaks back into the master document.

This thesis was partly funded by EU FP6-IST "EUPASS - *Evolvable Ultra-Precision Assembly SystemS*" (Grant agreement No: 507978). Other funding was received from the Tampere Graduate School of Concurrent Mechanical Engineering (GSCME) and from the K.F. and Maria Dunderberg foundation. I gratefully acknowledge the GSCME, under the leadership of Prof. Erno Keskinen, for both the financial and academic support provided. In the finalisation phase, some 'cents' were utilised from EU Hz2020-FoF-11-2015 project "ReCaM - *Rapid reconfiguration of flexible Production Systems through Capability-based Adaptation, Auto-configuration and Integrated tools for Production Planning*" (Grant agreement No: 680759), which also provides a platform for future work.

Finally, I want to thank my parents and whole family for believing in me and for giving unflagging support during my studies and helping me launch my academic career. I want to emphasise the role of my father, Ilkka, and my uncle, Eero Lehto, for sparking my interest and guiding me into the field of technology and automation, and to my mother, Sinikka, for raising

me to be the person I am. Lastly and most of all, I want to express my thanks to my lovely wife, Merita, and my daughters Veela and Aisla for filling my life with delight, with love, and for giving me the strength to continue. I hope they will forgive me for the long hours and days when my work has forced me to be absent from my home-life.

Niko Siltala
Tampere, June 2016

To Merita,
Veela and Aisla

Table of Contents

List of Definitions

List of Abbreviations and Acronyms

3D	Three-dimensional
API	Application Program Interface
AMD	Abstract Module Description
AutomationML	AutomationML (Description file format)
ASCII	American Standard Code for Information Interchange (7-bit character table. See ISO/IEC 10646-1)
BP	Blueprint (Description file format)
CAD	Computer Aided Design
CAEX	Computer Aided Engineering eXchange
CAN	Controller area network
CAPP	Computer Aided Process Planning
CIM	Computer Integrated Manufacturing
CNC	Computer Numeric Controller
CSS	Cascading Style Sheets
COLLADA	COLLAborative Design Activity
COTS	Commercially-available Off-The-Shelf
DB	Database
DH	Denavit-Hartenberg
DMS	Dedicated Manufacturing System
DOF	Degrees of Freedom
DOM	Document Object Model
DPWS	Device Profile for Web Service
DRM	Design Research Methodology
EDD	Electronic Device Description
EDDL	Electronic Device Description Language
EDS	Electronic Data Sheet
EAS	Evolvable Assembly System
Emplacement	Emplacement (Description file format)
EmplWS	Emplacement Web Service
EPS	Evolvable Production System
ERP	Enterprise Resource Planning
EU	European Union
EUPASS	Evolvable Ultra-Precision Assembly SystemS (EU FP6 project)
FDCML	Field Device Configuration Markup Language
FDI	Field Device Integration
FDT	Field Device Technology
FP	Framework Program (relates to EU funding instruments)
FMS	Flexible Manufacturing System
GUI	Graphical User Interface

GSD	Generic Station Description (Profibus related device configuration files)
HAL	Hardware Abstraction Layer
HC	History Container
HW	Hardware
HDLC	High-Level Data Link Control
HMI	Human Machine Interface
HMS	Holonic Manufacturing System
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IF	Interface
IEC	International Electrotechnical Commission
IP	Intellectual Property
IO	Input and Output
IoT	Internet of Things
ISO	International Organization for Standardization
JSP	Java Server Page
KPI	Key Performance Indicator
MAS	Multi-Agent System
MD	Module Description
MEI	Department of Mechanical Engineering and Industrial Systems at TUT
MES	Manufacturing Execution System
MID	Module Instance Description
MPS	Modular Production System
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
MVC	Model-View-Controller pattern for software development
NC	Numerical Controller
NPI	New Product Introduction
OMG	Object Management Group
OPC	Open connectivity via open standards (originally OLE for Process Control)
OPC UA	OPC Unified Architecture
OS	Operating System
PLC	Programmable Logic Controller
PLCopen	PLCopen - organisation
PLCopen XML	PLCopen XML (TC6 development of PLCopen)
PLM	Product Life Management
Profile	Profile (A description inside AMD. Instantiable by module)
PSL	Process Specification Language
RMS	Reconfigurable Manufacturing System
ROS	Robot Operating System
RQ	Research Question
SFC	Sequential Function Chart (a PLC programming language)
SI	International System of Units. from French *Système international d'unités*
SME	Small and Medium Enterprise
SMT	Surface Mount Technology (circuit board manufacturing process)
SOA	Service Oriented Architecture
SR	System Requirement
STEP	Standard for the Exchange of Product model data (ISO 10303)
SW	Software
SysML	Systems Modelling Language

TC	Technical Committee (relates to method of standard development)
TCP	Tool Centre Point
TR	Technical Recommendation
TRL	Technology Readiness Level
TTE	Department of Production Engineering at TUT
TTM	Time to Market
TTV	Time to Volume
TUT	Tampere University of Technology
TUT μFactory	TUT microFactory. Micro- and desktop factory concept developed at TUT/TTE
UI	User Interface
UML	Universal Modeling Language
UMRM	Unified Manufacturing Resource Model
UR	User Requirement
URDF	Unified Robot Description Format
URL	Uniform Resource Locator
URS	User Requirements Specification
VRML	Virtual Reality Modeling Language
W3C	WWW community
WS	Web Service
WSDL	Web Service Description Language
WWW	World Wide Web
X3D	eXtensive 3D (Successor of VRML format)
XDD	XML Device Description
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language
XPath	XML Path Language, is a query language for selecting nodes from an XML document.
XSD	XML Schema Definition, is formally describing the elements in a XML document.
XSLT	eXtensible Stylesheet Language Transformation, is a language for transforming XML documents into other documents.

Nomenclature

G	Global Frame
B	Body Frame
F	Interface Frame
T	Transformation matrix
O	Origin
P	Point
$\mathbf{r}$	Position vector
${}^{B}\mathbf{r}_{P}$	Position vector of point P expressed in coordinate frame B
$\hat{u}$	Unit vector
a	Denavit-Hartenberg (DH) link length
α	DH link twist
d	DH joint distance or link offset
θ	DH joint angle

Notations

`XML Element`	Elements (tags) of eXtensible Markup Language (XML) document, when they are referenced in text.
`XML Attribute`	Attributes of XML element, when they are referenced in text.
`URL`	Uniform Resource Locator (URL) to some external source in the Internet.

XML Path Language (XPath) [146] notation is followed defining the relations between XML elements and attributes. For example, '**`Element1`**/**`Element2`**@`attribute1`', defines that:

a) **`Element2`** is child element of **`Element1`** with use of '/' between these two entities; and
b) `attribute1` is attribute of **`Element2`** with use of '@' in the middle.

Definition 1: <Name_of_Definition>

Definition of a term or a concept worth to be highlighted, and to be explicitly defined.

1 Introduction

The traditional production systems used in Europe and other highly industrialised economies are struggling to cope with the challenges of modern industry due to their inability to respond quickly and efficiently to changes in demand. These challenges can be characterised as follows. The products themselves are getting smaller in size and increasingly complex, especially because of the increasing demand for customization and personalization, which always needs to be done as quickly as possible, if not by yesterday. At the same time, the lifespan of products is getting shorter and the markets are fluctuating ever more rapidly. Manufacturers need to 'make-to-order' instead of selling goods from stock, and this has increased the need for flexibility and adaptability in production systems. Nowadays, and for the foreseeable future, manufactured goods will be produced in ever smaller batches with more product variations, all of which requires highly adaptable production functionality and capacity. As a result of globalisation, unpredictable markets, the constant search for a competitive advantage and the demand for improved efficiency, modern industrial production and supply chains are characterised by a different set of requirements than in the past.

As the components of manufactured products get smaller and their quality increases, more precise handling and processing operations are needed. These place stringent demands on production systems. Smaller parts are more difficult to handle, not only for machines, but also for the human operators. In order to meet the customisation and personalisation requirements of customers. The increasing demand for manufacturers to offer made-to-order production operations means that batch sizes can be as low as one. This is already becoming common practice in many sectors, such as the automotive industry, in which cars are now routinely assembled according to an individual customer's pre-defined specifications. On top of all this, products need to get to market quicker, especially the completely new, highly-innovative ones, so the importance of Time to Market (TTM) and Time to Volume (TTV) are greatly increased [116]. These requirements impose pressure on manufacturers to speed up New Product Introduction (NPI) and all the associated ramp-up processes. Naturally, this increases the pressure on the production system designer to get the production system and all its constituent processes right first time; and of course, this all has to be done at a lower cost.

Many respected researchers have already pointed all this out. In discussing the requirements for next-generation manufacturing systems, Molina et al. [88] point out that, "The new generation of advanced manufacturing systems is forcing a shift from mass production to mass customization and the ability to manufacture in small batches. To achieve this, it is becoming increasingly important to develop modifiable, extensible, reconfigurable, adaptable, and fault-tolerant manufacturing systems. The reconfigurable and intelligent manufacturing systems are a solution for these market requirements." To achieve this holy grail of manufacturing, enhancements and improved support are needed in both a) system structures and architectures, and b) information and knowledge models representing the resources of those systems. It is the latter, i.e. formalising the information about manufacturing resources and the associated data models which is the focus of this thesis.

1.1 Technological Background and Motivation

We begin by defining a few core terms used in the context of this thesis. A couple of these have more or less established and agreed definitions in the literature, such as flexibility, which is defined as a system's property and capability to respond to changed conditions without the need for system reconfiguration [21, 70]. Flexibility is built in to a system a priori. The term reconfiguration is also well defined in the literature, being a means of changing the system's behaviour and/or layout by affecting its composition, i.e. a structural change takes place [21, 70]. This change can influence other parameters, Software (SW) or the physical entities of a production system.

However, there are other terms which are not so well defined, or at least are open to different interpretations. These include adaptability, which is the property of a system that enables it to be adapted to changed conditions. This can happen through parametric, logical or physical change [21, 65]. Then there is a system's agility, meaning how quickly and easily the system can be adapted or reconfigured to meet changed conditions. An interface here means a specified and documented connection between modules, while re-use refers to the re-utilisation of a working system's physical parts (modules) or SW in a new system. The idea here is that the modules should be like building blocks, which can be re-used in any new configuration of the production system without the need for any physical changes by utilising the existing interfaces for new inter-connections. Then there is granularity. The level of granularity defines at which level of a system's architecture the components can be changed. Another important term is fault-tolerance, which is a vital quality in a modern production system. If a system has high fault tolerance, then it is able to produce output at the required quality even if the inputs or processes vary. This can be achieved by means of, for example, adaptability. Finally, there is the data model, which defines the structure and semantics of a data set representing some phenomenon or entity in the system. In this thesis, it specifically refers to the digital abstraction or representation of a physical production module. More detailed definitions of some other terms, such as data, information and knowledge, and related production systems and paradigms are given in the literature review (Ch. 3). In addition, a number of other important terms are defined in detail in Ch. 4.2.1, such as exchangeability, interchangeability, interoperability, compatibility, and modularity.

There are many factors which have to be taken into consideration when creating a new production system for a manufactured artefact. Nowadays, sustainability and environmental factors always need to be taken into account when designing and implementing a production system. Government legislation and standardisation guidelines support this, and also set their own requirements for production systems. In addition, the production system itself may set its own requirements and limitations on the system's flexibility and agility. With any modern production system, it must be possible to adapt and integrate new technologies and technical solutions into any part of the system. [20] Another problem is the fact that most existing production systems have been designed traditionally for a specific product family, with little attention having been paid to the flexibility of the system [69]. Many existing production systems are rigid, highly integrated and tightly intertwined, and this greatly reduces the opportunities for the re-use and adaptation of the system, particularly in an agile manner. In fact, at present the most common procedure is to scrap a production system entirely once there is an NPI or some other change in the product requirements, and then to build a completely new production system from scratch. Obviously, this is highly undesirable from the point of view of environmental sustainability, and from most other perspectives too, not the least of which is cost.

There is little doubt that intelligent and easily-reconfigurable manufacturing systems are the way forward to meeting the increased requirements for adaptability, changeability, agility, fault-tolerance, and re-use [88]. The first prerequisite for an easily reconfigurable production system is that it needs to be based on the concept of modularity, i.e. a Modular Production

System (MPS). Secondly, it needs in-built mechanisms to enable rapid changes in configuration on a number of different levels. These include parametric, logical, and physical levels, [21, 65], capability and layout adaptations [65], and ideally, as many phases as possible should be completed automatically [34, 35]. Currently, there are a few industrial production systems which can boast ease of system reconfiguration, but this usually takes place either on the production line or at the work cell level [51]. This is too coarse a level of granularity to guarantee efficient re-use of all the production modules that may be available. The exchangeability level of a production module, i.e. its level of granularity, should be finer, meaning it can occur inside a production cell or unit. At present, this is only possible in very few cases in the area of machine tools. However, a finer level of granularity in the descriptions of production resources would greatly improve the agility of a production system, making it easier and quicker to reconfigure it through the efficient re-use of the existing (usually very expensive) production system parts.

This finer granularity can only be achieved by changing the system's architectures. The objective in such an approach is to be able to create complex and advanced production functions by combining atomic building blocks, each of which in itself may only provide the simplest of functions, such as 'grip' or 'press'. These atomic units have to be defined at a much finer level of granularity than they are today. This would enable the creation of more, and different, combinations of the atomic units, and would increase the reconfigurability of any production system built with these means. For example, the function of a production system could be altered by changing only one atomic unit, out of the many units which comprise the system. In practical terms, this could be the case when extending a range of one degree of freedom in a pick and place system. This is in contrast to present solutions, in which one large, monolithic production resource provides not only the required function, but a great number of other functions, too. In such cases, huge production units need to be completely replaced, often only in order to achieve one small change in one function, such as extension of a range. Another advantage of this concept of fine-grained modularity in production systems is economy of scale. The smaller atomic units which comprise a truly modular production system can be made in larger volumes, which would reduce the price per unit. A number of such reconfigurable production systems have been proposed and the main ones referred to in this thesis, Reconfigurable Manufacturing System (RMS), Holonic Manufacturing System (HMS) and Evolvable Production System (EPS), are defined and analysed in more detail in the literature review chapter (Ch. 3.3 (p.26)).

The enabler for these reconfigurable systems is information and knowledge management, which needs to be brought down to the same level of granularity as the physical resources which are to be exchanged. Therefore, the production resources must be represented comprehensively and in great detail. The digital descriptions of the production modules must include all the capabilities, the interfaces and the physical embodiments of the resources. Only in this way will production systems have the means and the tools to raise the agility and reconfigurability of a production system to the level needed to meet the challenges facing modern industrial manufacturing operations. Ch. 3.1.2 (p.20) of the literature review analyses the data and information required in more detail, and Ch. 3.4 (p.37) examines different existing data models.

Traditionally, the design of any product and its production system are tightly intertwined, and product requirements need to be created hand in hand with the system design. Britton and Torvinen [12] illustrate (Figure 1.1) how closely a product and its production system's life cycles are connected, from the initial design phase to the actual manufacture of the product. The figure shows how the physical intersection of product and production system takes place during the phases of ramp-up and manufacture. In practice, there may be several product life cycles which intersect in the same production system. In addition, [12] highlights the necessity of joint and collaborative design taking place at an even earlier stage of the manufacturing process. The optimum advantage can only be gained when both the product and its production system are designed in tandem, so that they fit each other's needs and limitations. What is proposed in this

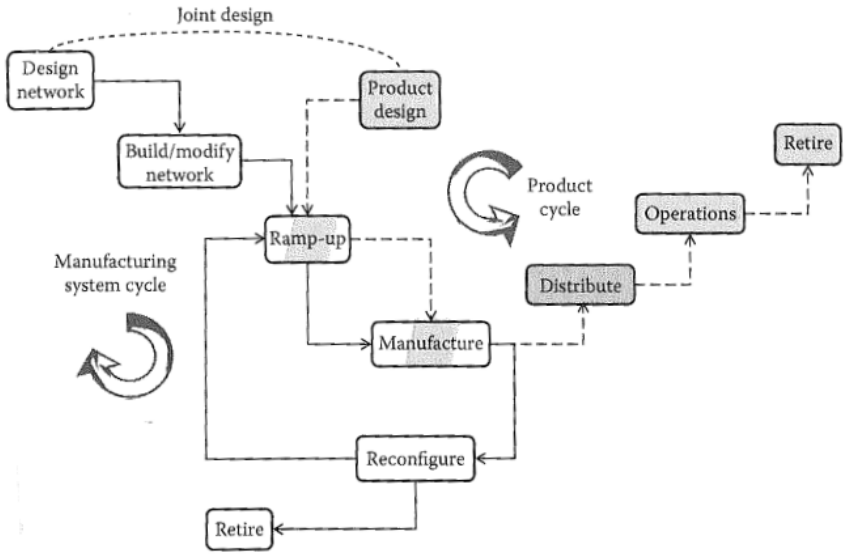


Figure 1.1: Harmonising product and manufacturing system cycles. [12, p.175]

thesis, however, is something different. What if this binding of product and production system design could be loosened?

Researchers into the relationship between Product, Process and Resource [106, 110, 116, 119] have suggested a method for postponing the binding of a product to its manufacturing system to a later stage in its development. In this method, the product requirements are matched with the resource offerings through the processes, which create the glue between the two. In this case, the product can be designed more independently from the production system, as long as common terms are used which later enable the product to be bound to the production system. It can even be bound to different kinds of systems, while still achieving the same end result and target quality. However, none of this is possible unless both the product requirements and the resources available for its manufacture can be described with standardised, shared terms used in a common format. Traditional approaches to production system design would also benefit from having the resource information available in a common, exact, and easy-to-share format, which would create a comprehensive, digital data sheet about a production resource. Bearing this in mind, the procedures for designing production systems with the help of processes providing the connection between the product and the resource specifications are analysed in more detail below in Ch. 3.2 (p.21).

A standardised format for resource description has many other advantages, too. At present, the range of tools available to design and implement production systems is very heterogeneous. Data models with a common format for resource exchange would greatly improve the integration of tools, and would streamline the development process for a production system. System designers and integrators could make better and faster decisions based on more thorough, correct, and valid information, which can be retrieved on-line. They would benefit immensely in terms of cost savings and quality improvements if they could get detailed descriptions of production resources in a common format. This would eliminate the need to recreate and retype information (often again and again) and would also lead to savings in time and a reduction of misunderstandings and errors. Digitally represented resource descriptions will also improve the quality of a production system, firstly by ensuring in advance that all the parts will fit together, and secondly, if the processes can

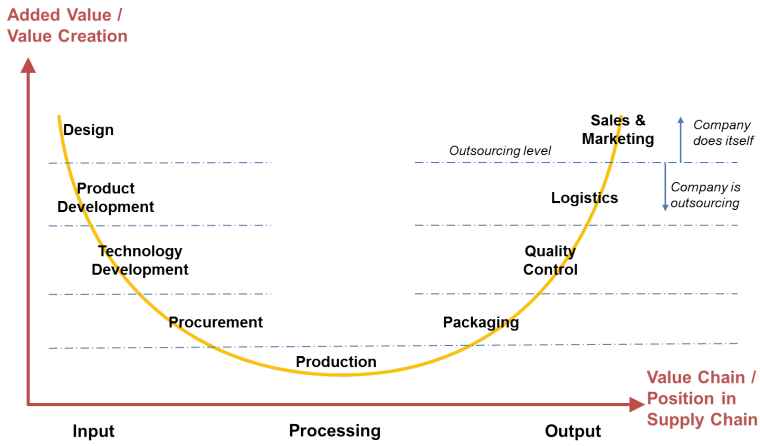


Figure 1.2: Smiley of the value chain and outsourcing levels. Modified from [68, 90]

be simulated digitally then it can be verified that the system meets the product requirements. All the necessary information for making these verifications should thus be available in the resource description. Given that it is the vendors of production resources who have the best, first-hand knowledge about their production modules, they would the best and most logical providers of precise, high-quality descriptions of the resources. Structured, detailed, and digital resource descriptions can not only reduce the amount of (technical) information requested from the vendor, but also provide new marketing channels for production resources.

1.2 Economic Environment

The economic environment for production systems is characterised by the reduction of production costs per unit while maintaining the required product quality, giving the manufacturer an edge over its competitors. On a broader canvas, the advantage of being able to understand a product's value creation from a production perspective is illustrated in Figure 1.2, the Smiley of Value Chain, proposed by Johansen and Vestergaard [68]. This topic has also been considered by other researchers [67, 91], who provide additional considerations as to where and how the production should be carried out. These clearly indicate that production is first in the line of fire when it comes to outsourcing. This is understandable, as building a production system involves high investment, high running costs and high risks. One of the greatest risks is related to market volatility, which increases the chances of a system becoming prematurely obsolete. On the other hand, the separation of design and product development from production has its own risks, too. When production gets separated from product design, the latter has a tendency to start designing products of poor quality from the Design for Manufacturing point of view, which can cause unnecessary additional costs in production. In the worst case, this can lead to the elimination of both production engineering and product development operations. This is of particular concern to companies making physical products, because there are direct links from the manufacture of a product's components and the assembly of the product to the design and value propositions of the sold product [91]. However, agility, reconfigurability and finer granularity of a system can be seen as countermeasures to this. With these features, the production system remains responsive to changes in the demands of the market and can be adapted to new requirements faster and in smaller increments. This also reduces the risk of an investment becoming obsolete. In a large industrialised market such as the EU, the aforementioned characteristics create the foundations

and for keeping the production in Europe, near to the R&D activities, which obviously benefits Europe's industrialised economies. However, none of the above characteristics can be achieved without common information models and detailed, consistent, and up-to-date information about the production resources.

The re-use of existing production modules in new or reconfigured production systems is beneficial for the maunufacturing industry. Harms, Fleschutz, and Seliger [43] and Weule and Buchholz [178] raise many of the issues and obstacles regarding the re-use of production resources. They list the main obstacles as being: a lack of knowledge and experience with the re-usable components, insufficiently designed and prepared equipment, insufficient information and statistics about the reliability of the re-usable component, missing life cycle documentation and the lack of holistic planning concepts for re-use within companies. [43] identifies the adaptation processes required to meet the required functional capabilities as the main cost driver for re-use. This raises the risks involved if there are unforeseen adaptations. However, formalised descriptions that provide detailed information about a module and its operational status would enable, improve, and create better opportunities for the creation of production systems made from available modules, both new and re-used ones.

As well as the costs of Hardware (HW), installation, and project management, SW development and ramp-up are major factors in the cost of a new production system . The losses in quality and delays in the ramp-up of a production system and in TTM caused by poor SW are highlighted in [63, 112]. The share of SW in modern mechatronic device projects, e.g. production systems, is increasing all the time. [112] states that commissioning control systems can take up to 22,5% of the entire time devoted to a project, and the major part of this (up to 70%) consists of debugging the software. Both [63] and [112] propose virtual commissioning as a solution for this issue. Additionally, [63] includes component-based aspects in its solution, enabling even more rapid development. However, if we do not limit ourselves only to the virtual world, then the same components could be implemented and applied more efficiently in both the digital world and in reality, which would lead to even more savings in time and an increase in quality.

All the above-mentioned issues boil down to information and knowledge, and how to store and access it. To date, production resources have not been described with complementary means and measures, i.e. in a common and standardized format in which information can easily be shared, exchanged, compared and evaluated between different players. This thesis will contribute to the realisation of this concept and will provide data models to describe production resources digitally, making future production system design and commissioning processes quicker, smoother and more efficient.

2 Research Description

This chapter has several aims, the first of which is to define the research problem, formulate the research objectives and pose the research questions. Next, the research methodology and research approach are explained, as are the assumptions and limitations affecting them. The chapter then defines the original contribution to the topic made by the author, and concludes with a description of the overall structure of the thesis.

2.1 Problem Description

In [46], Hollis and Quaid present the concept of an Agile Assembly Architecture, which they elaborate on in [45, 71]. It describe how a modular manufacturing system can be designed and re-used. Their scenario is represented in Figure 2.1, and includes the following main phases [46]:

a) Design with access of remote modules via the internet.
b) Complete the geometric and functional model.

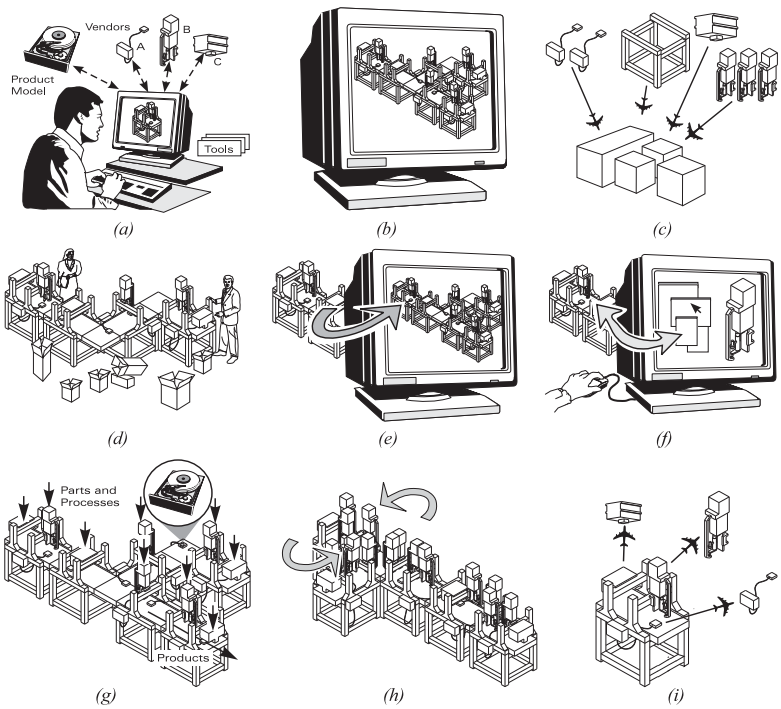


Figure 2.1: Scenario for Agile Assembly Architecture concept [46]

c) Order and ship the modules to manufacturing site.
d) Rapid assembly of the system.
e) Automatic generation of updated model from real system.
f) Interactive graphical programming.
g) Operation.
h) Rapid re-configuration to meet the changing requirements.
i) Re-use of the components no longer needed.

Even though this scenario was first presented some years ago, it has never become a reality, at least not in any broad or generally accepted sense. This is because their scenario lacks sufficient information about the constructional elements, i.e. the production system modules. The same applies to the Reconfigurable Manufacturing System (RMS) paradigm [21] and its descendants (See Ch. 3.3 and Figure 3.8). Because there is little comprehensive and formal information and/or knowledge about available production modules, the use of such modules in practical systems is inhibited, or even prohibited. This has led to a situation in which the full potential and advantages of RMS and its like cannot be harvested.

2.2 Research Objectives and Research Questions

The idea of increasing a systems' agility has been pursued in [1, 46], and this concept is the foundation for this thesis. This thesis aims to improve the scenario presented in Figure 2.1 by offering a formalised information model about the available production resources/modules for re-configurable systems. This information model can be utilised in various ways in all phases of the posited scenario. However, this work goes somewhat beyond that, as the objective is to provide abstracted models which can be used for module design, harmonisation, and exchangeability.

Thus, the aim of this thesis is to define a description format(s) and associated data models for production modules that can be utilised in the design and implementation of a variety of modular production systems. The module descriptions have to be presented in a generalised form, so that different kinds of production processes and modules can be expressed by the same descriptive format.

The **ultimate objective** of this thesis is to improve the design, implementation, and ramp-up procedures for modular production systems by improving their adaptability, agility, efficiency, and quality. The thesis not only aims to improve the deployment and commissioning process in order to make the reconfiguration of production systems more rapid and more autonomous, but it also aims to improve the sustainability of production systems through the re-use of the production system and, in particular, its constituent parts, i.e. the production resources/modules. This overarching objective can be achieved by developing a methodology, specifications and tools for producing detailed production module descriptions which will support computer-aided production system design and the configuration process. This main objective can be divided into the following sub-objectives:

Sub-objective 1: To define the requirements for production module descriptions

This first sub-objective is to collect and compile the user requirements which underpin any production module descriptions. This includes identification of different user roles associated with the resources and the system design. These user requirements are used to identify the needs, and thus to define the system requirements.

Sub-objective 2: To define the description concept for production modules

The aim is to support all the stages of a comprehensive decision-making process, from comparison and evaluation of the conceptual design to the finest detail of information needed to design an efficient and high-quality production system. This is not to forget the final target, which is to have a fully-operational production system up and running with minimum time and effort, and to keep it in running condition with minimum downtime.

Thus, the second sub-objective is to define a description concept that supports and facilitates production system design using predefined and existing modules in a heterogeneous multi-vendor environment. The concept should support current RMS and novel EPS-related system design processes. In particular, it should fulfil the needs of the different iteration phases involved in the design and commissioning processes of a production system.

A concurrent objective of this concept is to enhance the exchangeability and interchangeability of the modules in the production system. These are the enablers of a truly multi-vendor production environment that can cope with competition and cross-vendor upgrades. They enable fast changeovers and short system downtimes in the case of module breakdowns or process upgrades. A further aim is to refine the level of a system's granularity by at least one step over current production systems, i.e. to shift it from the inter-cell level to the intra-cell level. The modules described here are preferably elementary or atomic in nature, and can thus be combined into more complex entities. All these objectives lead towards improved sustainability for a production system, as the modules are intended to be re-used more easily, and thus should remain viable for a longer time.

Sub-objective 3: To develop and implement a format for formalised module description(s) which can offer a comprehensive representation of the production modules

The third sub-objective is to provide comprehensive and complete description(s) of all aspects of the production modules. The module description(s) given in this work can be utilised in various phases during the design, commission, and execution of a production system. Furthermore, they offer considerable savings in the time, effort, and money consumed during system design and ramp-up. In order to make optimum use of these description(s), they not only need to be in a computer-readable format, but also in a format that can be read and understood by a human expert. In addition, the module descriptions will provide formalism and harmonisation between the modules' origins, which may come from various sources. The advantages to be gained from a common description format and the consequent reduced effort expended on re-creation, retyping, and reformatting of information about the production modules are immense. Thus, the aim of this description concept is to enhance the accessibility of module descriptions and promote their easy and open availability. This can be achieved by creating a digital resource pool of production modules, i.e. a library of resource descriptions. Ideally, both the description concept and the description format(s) will support different modular production system paradigms, such as the RMS and EPS paradigms which are focused on in this work.

Research Questions

Based on the aforementioned objectives, the following research questions were formulated. The principal Research Question (RQ) 0 can be formulated as: **How to support and improve the design-by-reuse and module selection processes in designing and commissioning reconfigurable production systems?** This is divided into the following more detailed RQs.

RQ1. **How can a production system's resources be represented so that the needs of the different phases of system design can be met, along with the requirements of the identified users?**

RQ2. **What information about the production system modules needs to be collected and represented?**

RQ3. **How can the module descriptions be utilised by the different user roles in different phases of the production module's life cycle?**

RQ1 considers the overall system design process and the framework used to create the foundations for the representations. In addition, it considers how the proposed representation would support and contribute to this overall framework, and provide links between the different parts of the framework.

RQ2 investigates and evaluates what is the optimum number of module descriptions (e.g. different scopes, intended use, and level of information required), and what they need to contain in order to meet the identified requirements. Furthermore, it addresses how the (interface) standards and the processing capabilities of the production modules can be represented formally.

RQ3 also considers which tool set is required and how these different tools can be used for the benefit of the overall design framework.

2.3 Research Methodology

The following research methods form the methodological foundation for this research. First of all, the Design Research Methodology of Blessing and Chakrabarti [9] is utilised as the basic framework for clarifying the research problem, formulating the proposed solution and evaluating the results. This is supplemented by System Engineering [129] and case-based research [179] methods.

Design Research Methodology

Design Research Methodology (DRM) is used to develop the proposed methodology and concept. The same methodology was applied in the design of the formal language used to represent the production resources. The DRM is presented in [9], and, supported and complemented by the other described methods, it forms an overall methodological framework for this thesis. The DRM method has four distinct phases: Research Clarification (RC), Descriptive Study 1 (DS-1), Prescriptive Study (PS), and Descriptive Study 2 (DS-2). In this research, these four phases are implemented using the following methods:

1. **Research Clarification (RC)** At this stage, the goals of the research are formulated through an analysis of the relevant literature. The RC consists of a *literature review* supported with information gathered from *action research* and *industrial workshops*. The literature review mainly concentrates on manufacturing, production and computer science-related publications and journals; peer-reviewed conference articles and compilations; and books.

It was conducted with the dual purpose of describing the background and requirements for the study and recognising the state-of-the-art developments in the field.

2. **Descriptive Study 1 (DS-1)** This phase included an extended analysis and review of the literature. In addition, empirical data was collected to provide a deeper understanding of the research problem and the factors influencing it. DS-1 also included an investigation of the relevant industrial and commercial *standards*, although most of the information came from *workshops and interviews*. Later, workshops were used to collect information and iterative feedback on the developed concept and the information models. These work in parallel as validation and relevance checks for the developed concept and the specifications. During this phase, the information gathering was supplemented with a *questionnaire* used to gather information about the de facto (interface) standards applicable in the field. These were used as one of the cornerstones of the description concept.
3. **Prescriptive Study (PS)** In this stage, all the experience, knowledge and assumptions about the problem were synthesised into a proposed solution. The PS contains the main development operations and utilizes the methods of *System Engineering* [129] and *action research*. These were applied to the development environment in order to develop the generic models and the formal descriptions of the production resources.
4. **Descriptive Study 2 (DS-2)** During this final stage, the proposed solution was analysed and evaluated against the defined goals using empirical data. The *case-based research* [179] method was utilised to evaluate the results of the validation case(s). This was applied to two separate cases, the first of which is more focused on the development of the concept, while the second focused on validating the results of the developed concept and the formal descriptions. The case environments are also used to demonstrate the utility of the proposed concept.

Research Approach

Utilising the aforementioned research methodology, the objectives of this thesis were approached with the following steps and associated methods. The overall research approach is depicted in Figure 2.2. The boxes with the 'S<#>' designations connect the graph with the steps presented below. The following main steps in the overall approach can be isolated and highlighted. These will lead to the results of the research questions and objectives:

Step 1: Collection and analysis of the requirements and boundary conditions (RC, DS-1)

This first step focused on the collection and analysis of the requirements and boundary conditions for the description concept. This step includes: the identification and analysis of the production system design and reconfiguration process, identification of the key stakeholders and their requirements for the concept, and identification of the use-case scenarios. These use-case scenarios define where and how the concept can be applied. In addition, the requirements and implications arising from the execution phase of the production system are analysed. The gathered information is presented in the literature review, and is further formulated to support general, user and system requirements.

Step 2: Development of the methodology and concept (PS)

This second step develops the concept for describing the production modules digitally, and the associated methodology. The developed concept and the Generic Model should represent a comprehensive description of all aspects of a module's functionality, including all the implications from the product side, the process side, and from the interaction with other modules. These

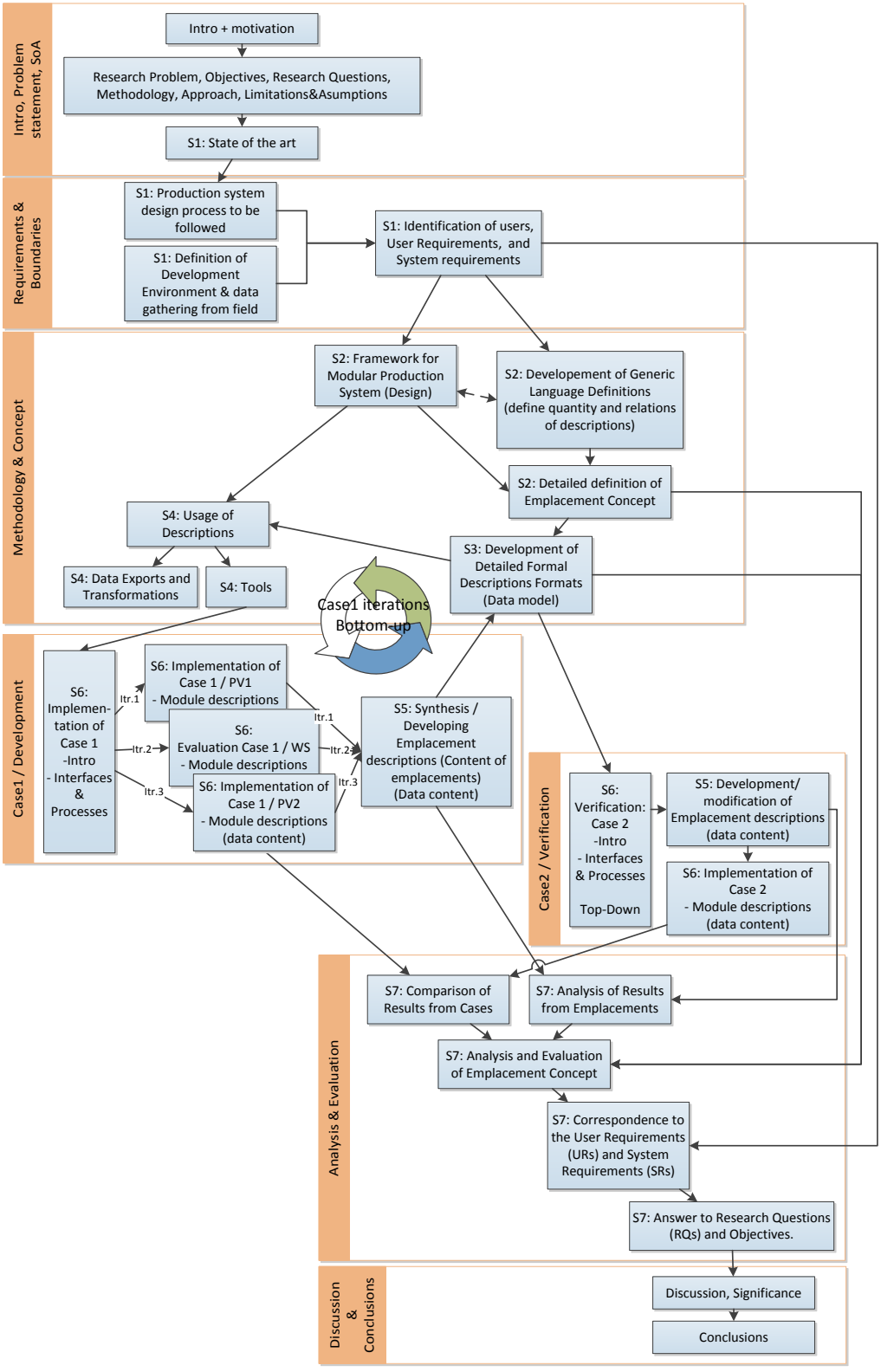


Figure 2.2: The overall research approach

include all the factors related to the geometrical, mechanical, electrical, and functional points of view.

The developed description concept should recognise the specific requirements of the different phases associated with the design and implementation of a production system, as found out in step one. A (design) framework is described which reflects the different phases that occur during production system design. This shows that a multi layered concept needs to be developed, which will enable and guarantee harmonisation and interchangeability between the modules, and provide mechanisms to create designs and make informed decisions using the appropriate level of information. The concept should work on at least two levels, the first being higher and more abstract, and the second more practical and concrete, i.e. the real module level. Action research and systems engineering methods are used to carry out this step.

Step 3: Design of a formal language(s) to describe the production modules (PS)

The third step is to design a formal descriptive format(s) which fulfils the requirements of step one, and fits the description concept and Generic Model developed in step two. The developed format and underlying data models should be generic so that they are capable of representing the different kinds of processes and modules which exist in the production domain. The included information relates to the module's characteristics, properties, interfaces, process capabilities, and capacities. Action research and systems engineering methods are used to perform this step.

Step 4: Identification and development of supporting tools (DS-1, PS)

The fourth step focuses on the detailed identification and use of the developed concept and associated tools for the formal language(s). The intended uses for the descriptions are identified during the collection of the user requirements. This step includes the development of a few key tools and processing methods for the developed descriptions. These are mainly used for developing, processing, and sharing the descriptions. One such is the implementation of a tool for distributing and processing the module description files over the internet. Iterations and frequent updates are needed, as the tools need to reflect and be compliant with the current versions of the description's data models. Systems engineering methods are used to carry out this step.

Step 5: Development of the Emplacements and Identification of the required interface standards and capabilities (PS)

The fifth step is collecting the information needed to fill in the data models developed in step three. This defines the content of the module description and includes the formalisation of the interface standards and capabilities. First, the applied (interface) standards and capabilities need to be identified, collected, and formalised in order to fit into the developed data models. Next, the Emplacement descriptions are filled in with the relevant data. This step also includes the development of a method for managing and using the interface standards as grouped packages.

In the first phase of development, the iterations rotate from the bottom up by analysing the development environment and defining the module descriptions. These are synthesised into Abstract Module Descriptions and updates for the data models, followed by the step four tool updates. This iteration is illustrated by the looping arrows in the centre of Figure 2.2. In the second phase, when the data models are established and the research is approaching the validation phase, the iterations change to top-down. The Abstract Module Descriptions are first updated to reflect the specifications of the new production modules. The tools are used to develop templates for the Module Descriptions, after which the production module data is added. The main difference in this phase is that the data models and tools should remain untouched. The action research method is followed for these phases.

Step 6: Validation of the results (DS-2)

In the sixth step, the results are validated. The two case studies are used for verification and validation of the developed description concept and the associated methodology, the developed Generic Model for the module data, and the data models used for the formal module descriptions. The developed module descriptions, i.e. the digital representations of the modules, are also analysed. In the first case, the production modules of the EUPASS line are used simultaneously for development and verification purposes. In the second case, the production modules of the TUT μFactory are used as the validation case. Case-based research methods are applied to the case studies.

Step 7: Evaluation of the proposed concept and descriptions (DS-2)

In the final step, the proposed concept and descriptions are analysed and evaluated against the defined requirements and objectives, after which some valid conclusions are drawn from the research.

2.4 Assumptions and Limitations

The following assumptions and limitations have been made for this thesis. Firstly, the proposed concept is intended to be used in the context of Modular Production Systems, and more specifically in the context of paradigms such as RMS and EPS. It is assumed that the target system is composed of modules, and that these modules are exchangeable. Thus, all the applicable production systems must be based on the concepts of MPS, which are implement first, followed by their reconfiguration characteristics. The main perspective is that of the System Integrators and Module Providers. The proposed concept and associated description formats are intended to be a vehicle for the different tools and applications to exchange information about the production modules. The applications can be system design tools and frameworks, ontologies and execution frameworks. The sources of information for this concept are the physical modules, the process capabilities and the (interface) standards. All of these need to be represented in a digital form. Thus, this work is limited to the Interface Control Document [129, pp. 61-63] existing between the different applications, rather than system development, per se.

Secondly, the framework is intended for the design and commissioning of production systems. This is a basic premise for the proposed concept. In the framework, although a lot of tools are identified, they are (and will be) missing for now, as they are beyond the scope of this thesis. The concept and its associated framework are not information-consuming applications themselves. They do not utilise the information, but are used to harmonise, store, share and distribute it. The framework offers a broader view of a larger (design and commission) system into which the developments made here are embedded and applied, as part of the whole.

Thirdly, the phases in the system design process are required to navigate the path from product design and requirements to a fully functioning production system. However, the product and system design, and the associated process phases are not considered in this thesis. This exclusion also applies to the information exchange formats and the tools required for such processes. For example, product specification, product-process requirements, system layout, and the recipe for a product are all beyond the scope of this thesis (See Figure 5.10 (p.98)). Because of this, an industrial case for the usage-based validation of the developed concept and the description formats is not feasible at the moment, and therefore excluded from the scope of this thesis. The validation is based solely on a manually performed analysis and the predicted benefits of the models, as well as a heuristic deduction of the expected benefits. Verification and validation of the cost modelling and agility potential of the proposed concept are also excluded. Therefore, only the necessary and

relevant parts of the system design process and the framework are presented 'as is', and these are not evaluated.

Connectibility can be seen as an issue on two different levels, the module level and the system level. The system level is higher, and there the interest lies in how and where the different components are connected together according to a layout specification. This level is beyond the scope of this thesis. The module level is the enabler of the interconnection and thus provides information for the higher level. It is this level of the module descriptions which is under consideration in this thesis.

Fourthly, this work considers the resources for production systems and how they can be utilised during the production system design process. Furthermore, this work focuses solely on the module level and defining the properties of those production modules. The module platform or Module System (including partitioning logic, architecture, module sets, configuration knowledge, etc.) [102, Fig. 4.1. p.172] are beyond the scope of this work.

Fifthly, this thesis does not take a position on a reference architecture, or any other architecture to be followed, as these are also excluded from the scope of this work. Even if the proposed concept and interface definitions assume that an underlying architecture does exist, any such architecture is assumed to be open, so the proposed concept is not limited to any architectural presuppositions. The concept and production modules described in this work may be used on any suitable architecture. However, the module designs and the internal selections for the modules are always made on the assumption that there is an architecture and a selected design rationale.

2.5 Contribution

The most important of the author's own publications related to this thesis are [121, 123–126]. These deal with the developed concept, called the Emplacement Concept, and the associated resource description file formats. They describe how the resource descriptions can be utilized, and an Emplacement Web Service for sharing, distributing and processing the production module descriptions. In addition to these five publications, the following original contributions were developed during the course of this work:

1. A production module description concept and method, called the Emplacement Concept, for defining the abstract and concrete production module descriptions.
2. The main contributor and developer of description file formats (data models in XML Schema Definition (XSD) format), and corresponding standards defining their content. The three formats are called Emplacement, Blueprint and History Container. The first two are standardised as EUPASS organisation standards [25, 26] mainly by the author, who has contributed the overwhelming majority of the ideas and work.
3. A Web-based tool and framework, called the Emplacement Web Service, which was developed for sharing, distributing and processing files associated with the proposed Emplacement Concept. See [126] and Ch. 7.3.1.
4. Creating 11 abstract level resource descriptions, i.e. Emplacements, containing 35 implementable Profile definitions. These are used for abstracting the production modules which appear in the case environments.
5. Creating 25 real module descriptions, i.e. Blueprint definitions, used in the two case environments for verification purposes.
6. Development of interface standards related to the case environments and the Emplacement Concept framework. These include technical contributions to the following standards: one International Organization for Standardization (ISO) standard [61] ($\approx$10 % contribution); three Evolvable Ultra-Precision Assembly SystemS (EUPASS) standards [22–24] ($\approx$25 % contribution for each); and three TUT μFactory standards [132, 133] (33 % contribution for

each) and [131] (90 % contribution). Apart from the ISO standard, the author has been the main contributor for standardising and documenting all of these interface standards (90 % contribution).

2.6 Structure of the Thesis

The thesis is structured according to the outline and structure shown in Figure 2.3. Chapter 1 introduces the research topic and presents the general problems. Chapter 2 refines the research problem and objectives, and formalises these into a set of research questions. Further, it discusses the utilised research methodology, defines the assumptions and limitations of this research, and identifies the author's contribution. Next, Chapter 3 reviews the literature and provides a few core definitions. This chapter has three major sections: the production system design process, targeted production systems, and existing description languages. Chapter 4 collects the requirements for the developed 'Description Concept'. Next, Chapter 5 introduces the developed description concept, called the 'Emplacement Concept', and defines the generic information models which comprise the concept. It also presents the framework of how the Emplacement Concept can be applied in production system design. Chapter 6 provides an implementation of the generic model in practical terms, by providing a set of eXtensible Markup Language (XML) based information models. Next, Chapter 7 discusses the utilisation of these developed information models. It discusses the use-case scenarios, the developed tools, and Case 1, in which the physical production resources are modelled and described with the proposed description concept. Chapter 8 discusses the verification and validation of the concept through Case 2. This is followed by discussion about the evaluation of the concept, its impact and significance, and future actions in Chapter 9. Finally, the conclusions of this research are presented in Chapter 10.

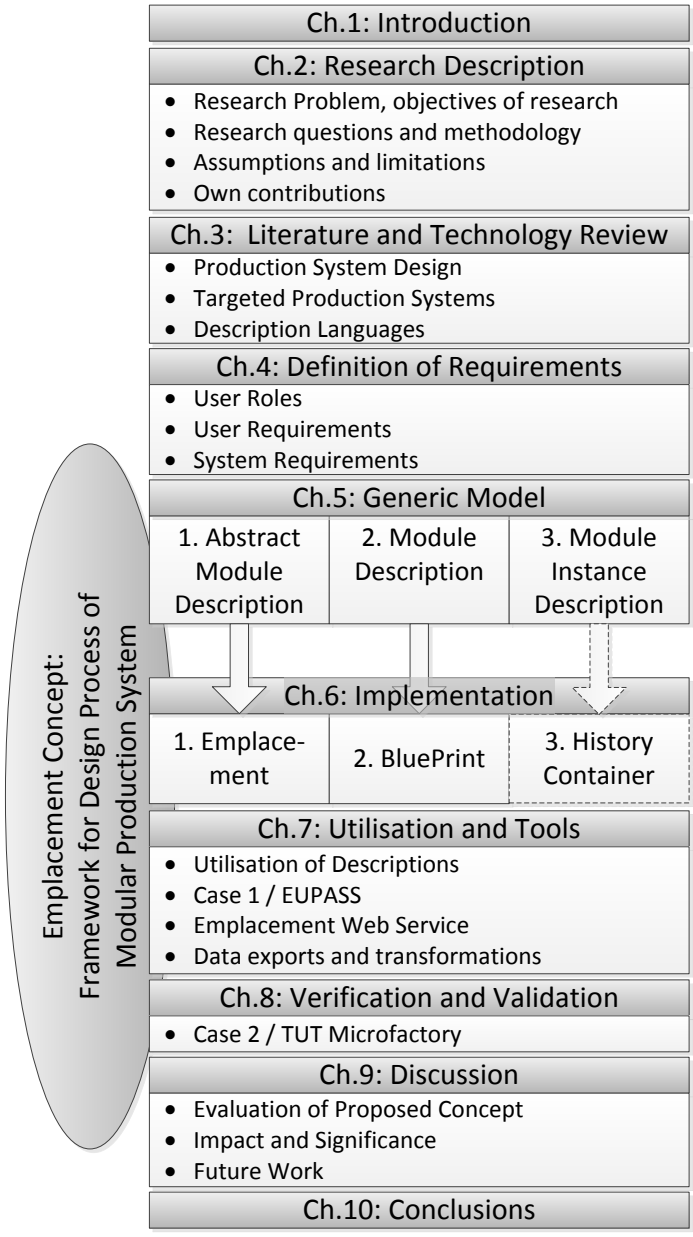


Figure 2.3: Structure of the thesis

3 Literature and Technology Review

The literature review collects together and introduces the literature pertaining to the basic concepts, a few of the central paradigms, the description languages and the technologies which are central to this thesis. The chapter begins by defining a few of the key concepts behind this work, and their associated terms. Next, the major issues concerning production system design (in the context of this thesis) are highlighted and discussed. Then the most relevant production paradigms are presented, which create the foundations and define a set of fundamental and general requirements for the presented concepts. Next, the competing candidates for the description languages, the module descriptions and the process descriptions are presented and analysed. The chapter also covers the literature related to controls and logics, Three-dimensional (3D) systems and kinematics, and User Interfaces (UIs). The different description languages are summarised at the end of this section. Finally, other research associated with the topic of this thesis is presented and discussed.

3.1 Introduction to the Key Concepts

This section defines a few key terms and concepts, and their relationships, in order to introduce the models and the interpretations followed in this thesis. This is done for the sake of clarity and unambiguity, as there exist conflicting views on the use of these terms and concepts. The first three terms to be defined are production, manufacturing and assembly, followed by the terms data, information, knowledge and wisdom.

3.1.1 Production, Manufacture and Assembly

In Europe, production is the most general and broadest term out of these three. It is the highest level category and encompasses both manufacture and assembly. The author's own view of the relationships between production, manufacture and assembly are best represented in Figure 3.1. Production is the aggregate of manufacture and assembly, and that is the sense in which the word is used in this thesis.

So, in the context of this research, production is, "The processes and methods used to transform tangible inputs (raw materials, semi-finished goods, sub-assemblies) and intangible inputs (ideas,

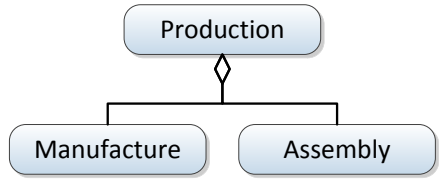


Figure 3.1: Relationship of production, part manufacture, and assembly

information, knowledge) into goods or services. Resources are used in this process to create an output that is suitable for use or has exchange value." [147, term: production]

Manufacture is an aggregated part of production. [148, term: manufacture] defines it as: a) The process of making something (wares) from raw materials by hand or by machinery especially when carried on systematically with division of labor, or b) "a productive industry using mechanical power and machinery". Manufacture is generally associated with processing, or the creation of parts or components from raw materials, blanks or semi-products. Examples of basic manufacturing techniques are milling, welding, moulding and painting. Usually, the process adds value to the product.

Assembly is also an aggregated part of production. In this work, the definition of assembly from CIRP [15, term: 1.0.2 assembly] is used: "The action of bringing individual parts into a sub-unit, a unit, a structural group, a machine, or a product. Note: 'Assembly' also includes the subsidiary functions of materials handling, adjusting, and inspection of the parts, sub-units, or final product. The result of carrying out assembly is a product, a unit, or a group of parts which is also commonly called an assembly." Generally, assembly relates to the process of joining together parts which have been created by the processes of manufacturing. That is the essential difference between manufacture and assembly.

From the production system's point of view, there is no great difference between a system used for manufacturing and one used for assembly. The production system is composed of connected production modules, and the control systems which control them. The descriptions and concepts produced in this work do not require any distinction to be made between these two, as the concept applies equally well in both domains. The basic procedures used to design production systems are the same. The difference lies in the variety of processes used, and the parameters implemented in the system, such as different production strategies and material flow. In the context of this thesis, production can be used as general term which refers to any or all of these three. Similarly, if the term '*X Manufacturing System(s)*' is used, it can equally well be substituted with either '*X Assembly System(s)*' or '*X Production System(s)*'. However, it should be emphasised that these terms are only interchangeable in the context of this thesis, which is concerned with how to create modular production systems.

3.1.2 Data, Information, Knowledge and Wisdom

The hierarchy and relationships between data, information, knowledge and wisdom are discussed in [8, 74, 115, 180]. All of these sources agree on the hierarchy and the order for the definitions. Basically, the data forms the foundation of a pyramid, and the other constituents are placed on top in the order of information, knowledge, and finally, in the top row, wisdom. This is illustrated in [115, Fig.1. – Fig.3., pp.164–168]. Although there are many differing definitions for these terms in the literature, there are no generally accepted exact definitions for these terms. What does appear to be generally accepted is that the higher level terms are usually defined in terms of the lower level ones, which is the pattern followed in this thesis.

Data (singular, datum) is raw, unprocessed information or objective facts. It is symbols, characters, values or images recording activities or situations, which tries to capture the true picture or real event. It does not have value or meaning of itself and thus it knows nothing. Normally data is historical. [8, 74, 115, 180]

Information is an organised and structured collection of data [180]. Information is data that have been given a meaning and structure by providing relational connections and a context. It is data that have been processed and are intended to be useful. It provides answers to questions like "who", "what", "where", and "when". [8] Another aspect of information is that it is the input for a decision or action, as is pointed out by [74], who summarises the essence of information

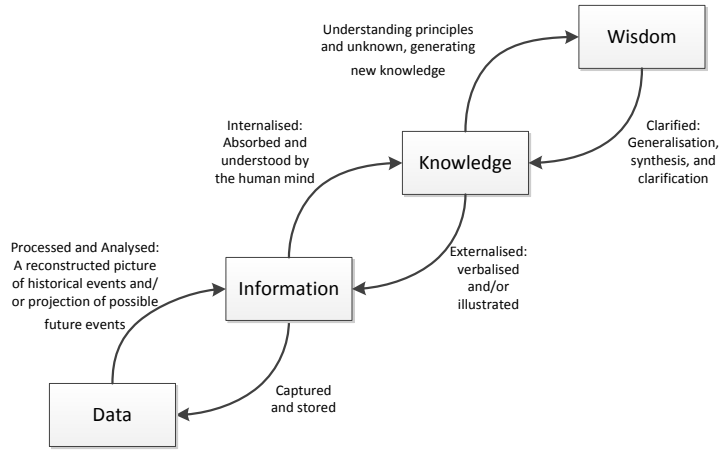


Figure 3.2: Relationships between data, information, knowledge, and wisdom. Modified from [8, 74]

as follows: The purpose of information is to aid in making decisions and/or solving problems or realizing an opportunity.

Knowledge is the application of data and information. It answers the "how" questions. Knowledge is the appropriate collection of information, that is intended to be useful for something or someone and it contains a pattern which is followed. [8, 180] Bellinger, Castro, and Mills [8] define knowledge as, "... a pattern that connects and generally provides a high level of predictability as to what is described or what will happen next". Liew [74] defines knowledge as "the (1) cognition or recognition (know-what), (2) capacity to act (know-how), and (3) understanding (know-why) that resides or is contained within the mind or in the brain." He summarises that the ultimate purpose of knowledge is value creation.

Wisdom is even less tangible, and is related to knowing "why" and evaluating understanding. It is something that is present at individual and organisational levels [115], but does not apply to machines. As such, it is beyond the scope of the descriptions presented in this thesis.

The relationships between the aforementioned terms are represented in Figure 3.2. The concept and descriptions presented in this thesis focus mainly on the level of *information*. However, this work inevitably refers to features from both the adjacent levels, i.e. data and knowledge. *Data* is collected and captured from the production modules in a raw form and is processed and analysed into information. *Knowledge* is captured inside the models in certain limited areas of the concept and descriptions, but exists mainly in the applications and tools which utilise the proposed concept.

3.2 Production System Design

Production system design is presented as one of the foundations for this thesis. It contributes to the requirements in the presented framework, which in turn leads to the proposed concept and the description formats. Indeed, the advent of comprehensive module descriptions will have significant effects on production system design processes, and this will be much discussed in the course of this work.

In order to achieve the objectives presented in the introduction, a modular and reconfigurable approach to production system design is needed. Such a design process is discussed below, where the close relationship between product, process and resource is highlighted, and its implications for the system design processes and the resource descriptions.

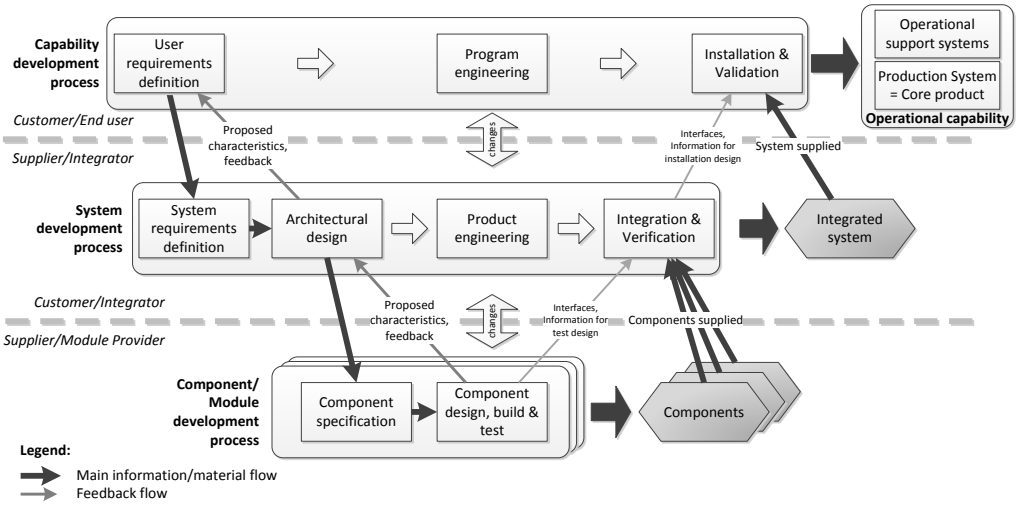


Figure 3.3: The production system development approach. Composed from [129, Fig.8.7–8.10]

3.2.1 The Production System Design Process

The overall goal of the systems engineering process [81, 109, 129] is to provide an operational production system and production capability, which are in turn defined by the business objectives. There are two distinct sided to providing a production system with the desired operational capability. One side specifies the required capability, i.e. the product requirements, while the other provides the resources needed to implement the specified capabilities. These two sides need to be brought together and so that they match each other in terms of the engineering processes. Three distinct development processes are required to achieve this: [129, Ch.8]

1) The capability development process aims to deliver what the users want by defining user requirements. It installs and validates the system as being ready for operation, and finally delivers the operational capability.
2) The system development process develops a system to provide the required capability. It responds to the user requirements by producing system requirements, followed by an architectural design. That involves developing or re-using components, and specifying interactions between the components. Finally, when the components are ready, the system development process integrates them and provides a tested system for the development process. This development process may be replicated and repeated N-times, as a separate sub-system development process, as an inter-system development process, and as a component development process. The output will be either a complete production system, or an integrated sub-system.
3) The component development process develops, constructs, and delivers a tested component ready for integration. Each component require a separate instance of the component development process. However, the intention, especially in this thesis, is to have the components ready for re-use in any reconfigured or new systems.

The complete approach to the production system development process is illustrated in Figure 3.3. The components and principles of V-model [129, Fig.1.5, Ch.5, Fig.5.3, Ch.6.4] [81] are also followed here.

The basis of the proposed concept and the key assumption in the production system design process is the tight relationship between product, resources and process. This will be elaborated on in more detail below.

3.2.2 The Relationship between Product, Process and Resource

Rampersad [109] presents an Assembly Model that is a concentric design model used for designing assembly systems. In this model, parallel and continual interactions follow each other, moving between the three main categories of product, assembly process, and assembly system. These categories are further divided into three levels of abstractions as illustrated in Figure 3.4. The entire set of items are termed 'assembly variables'. The inner level of the abstractions, which includes the product components, the assembly operations and the system components represent all the elementary parts, operations and (sub)systems. The middle level shows the product, assembly and system structures, and illustrates the mutual interrelationships between the components or the operations. The outer abstraction level defines the product variants, the choice of alternative assembly methods (strategy), and the proposed spatial positioning of the system components (layout). The arrows represent the various relationships between the previous variables. The thicker the arrow, the tighter the relationship.

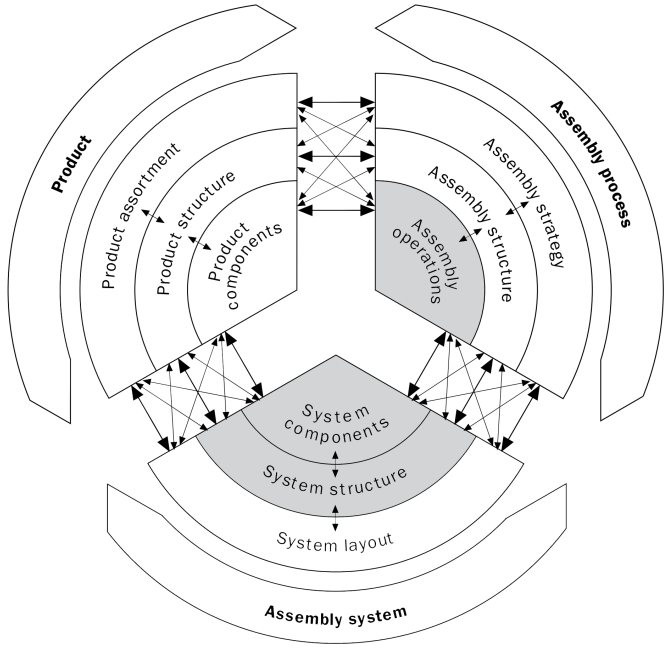


Figure 3.4: Assembly Model by [110, p.2]. Grey indicates the focus of this work.

Rampersad [109] points out the strong coherence between different variables in his Assembly Model. A change in one or more variables causes changes in other variables. For example, if the product is characterised by a set of properties, such as stiffness, size and weight, these will have a direct effect on the available assembly process parameters, like what kind of feeding, gripping or handling operations can be utilised and what degree of complexity is involved in the actions. This, in turn limits the available options for system components. This relationship is captured and formulated into equation: [109]

$$Assembly\ process = f(Product,\ Assembly\ system) \tag{3.1}$$

The design of an assembly system usually requires consecutive iterative loops. The iterations start from the outer ring of abstraction and spirals steadily inwards to the centre of the Assembly

Model, ending up in the operational system. Rampersad's paper presents and emphasises the importance of methods for discovering alternative systems. The iterations have decision points, where competing solutions may be evaluated and alternative routes taken. A full description of the iteration loops is presented in [109, Fig.7].

In principle, the cycle can be reversed [65]. In this case, the existing production system is taken as the starting point for the iterations. The system has a specific layout and possesses certain capabilities. These capabilities are actually the available assembly operations of the existing system.Whenever possible, this is taken as the initial point for product design, or more commonly redesign. Otherwise, the analysis of missing assembly operations will trigger a request for reconfiguration of the production system.

In addition to [109], there are a few other papers which have discussed the relationship between product, process and resource. These are discussed below, with the focus being on the ones that use the process as the glue, or link, between the product requirements and the resources capable of producing those products. Maropoulos, McKay, and Bramall [82] talk about resource-aware planning, by which they mean the creation of dynamic relationships between the planning entities and the enterprise resources, both humans and machines. The relationships are created in the process model by linking the features of the product to the resources, particularly at the enterprise and supply network level. Their study contains number of relevant points for this thesis, such as their model of the overall distributed planning functionality [82, Fig.1]. Cutting-Decelle et al. [18] collect together different definitions made in international standards for each of the terms: product, process and resource. They present the key features of ISO 15531 MANDATE standard, and discuss how it can be used to link product requirements to resources. The paper highlights the importance of defining information models and fixing on terms that are both commonly shared and agreed upon,i.e. standardised. Sandin, Gröndahl, and Onori [116] first present a process-oriented product design concept, which is based on an Assembly Module Platform (AMP). A standardised set of production processes are used as guidelines to design the products. Then, the AMP modules are created to implement the very same processes. Thus, the link between the product requirements and a module providing a technological solution can be created through the process-oriented concept. Shabaka and ElMaraghy [119] discuss how product features are analysed and then mapped to the capabilities of Reconfigurable Manufacturing System (RMS) machine tools in the machining domain. This mapping process leads to the generation of suitable machine tool structures to meet the requirements of the product. Horbach et al. [47] present a method for component-based planning, which has the components in two different domains. The first is the object domain, which represents the production processes, while the second is the method domain, which represents the planning process. These two processes are then combined. Pfrommer, Schleipen, and Beyerer [106] define a taxonomy to represent skills and tasks in terms of product, process, and resource (PPR) and their relations. The concept is illustrated in Figure 3.5.

This thesis focuses mainly on the information perspective of Rampersad's Assembly Model. Specifically it focuses on the Assembly System sector, but it touches partially on the information needs of the Assembly Process sector, particularly the abstraction layer of Assembly Operations (See Figure 3.4 and areas marked with grey background). The layers of Assembly structure and System layout will also be discussed.

The relationships between product, production process, and resources are summarised in Figure 3.6. The starting point is the product requirements and the production system resources. The production processes create a link between these two. The production processes are used to map the product requirements with the offerings of the production resources, which can be called capabilities or skills. Once a satisfactory configuration and layout of resources has been identified, they establish a plan representing the partonomy of the aimed-for production system.

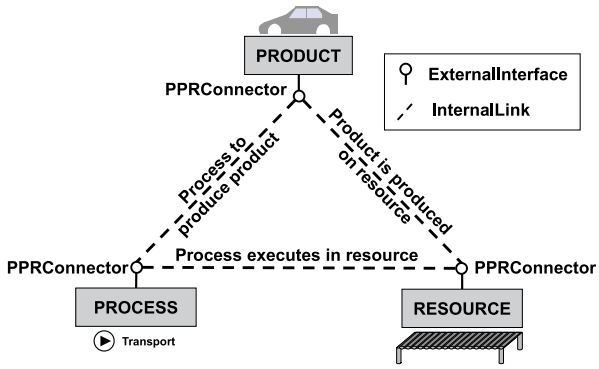


Figure 3.5: Product, process, and resource (PPR) concept [106, Fig.1.]

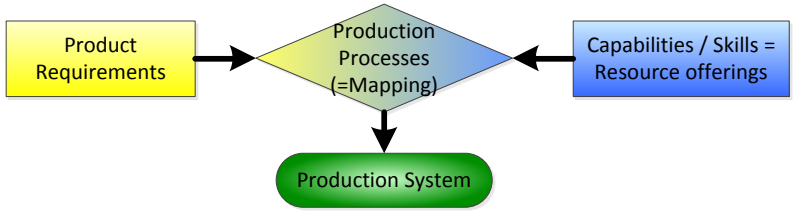


Figure 3.6: Relation of Product, Production Process, Resources, and Production System

This thesis focuses on the realisation of the right-hand branch of the figure, starting from the resource offerings, and moving through the process mapping to the production system.

3.2.3 Granularity

Granularity is an indispensable term used in defining the proposed concept and the associated module descriptions. The finest level of granularity defines the finest level at which the modules can be exchanged in a system. It is also of great importance when designing the production system. The selected system architecture dictates the level of modularity and the modules' exchangeability, thus determining the granularity of the requested system. A system's granularity, and much of the terminology, is defined as 'Equipment hierarchy' in standard ISA S95 / IEC 62264-1 [51, Fig. 4]. However, the fine level of granularity aimed for in this thesis is beyond the level specified in that Standard.

The reason for going into the finer levels of granularity has been stated by Sandin, Gröndahl, and Onori [116] as follows: "To achieve the full level of flexibility, assembly solutions must be designed to integrate any form or type of equipment: truly standardised interfacing. The equipment, in turn, must be broken down into smaller, process-oriented components."

Lohse [75, Fig. 6.15] and Lohse, Hirani, and Ratchev [76] propose a model going further towards finer granularity. Figure 3.7 combines the above-mentioned granularity hierarchies and terminology. For example, site is from the aggregation of one or more areas, area is an aggregation of a production line(s) and/or its work cell(s), and so on. It represents the different entities comprising a system, and at the same time the different possible levels of granularity. This model and the terminology it uses are followed throughout this thesis. The reconfiguration level of current systems normally ends at the cell or station level. However, the objective of this thesis is to bring the granularity level down from this, to the unit and element levels. As this thesis focuses

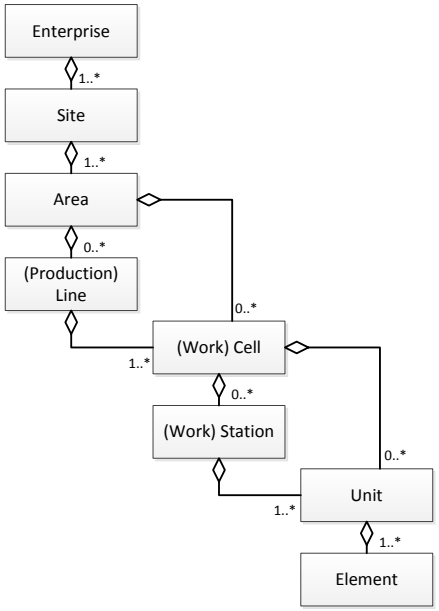


Figure 3.7: UML class diagram representing the component hierarchy of production system. Synthesised and modified from [51, 75, 76]

on module descriptions, the developments in this thesis aim to be applicable for granularity levels from the level of the Work Cell down to the Element level. The target granularity level should be described in the architecture specifications and the specific module descriptions.

3.3 Targeted Production Systems

A literature review of the targeted production system paradigms is presented in this section. These establish the foundations for the proposed approach. The emphasis is placed especially on the Reconfigurable Manufacturing System (RMS), the Evolvable Production System (EPS) and the Holonic Manufacturing System (HMS) as these are the main paradigms which might potentially utilise the module descriptions of the proposed concept. EPS, in particular, will be discussed in more detail, as will Dedicated Manufacturing System (DMS) and Flexible Manufacturing System (FMS), which are presented as points of comparison to the first three ones. However, the proposed descriptions could also be utilized for FMSs.

Different system concepts have always been developed over time to overcome the challenges for manufacturing systems. The early industrialists, like Henry Ford, started from the earliest manifestations of automation and went on to create the assembly-line model of production, which is still adhered to in manufacturing today. Their dedicated manufacturing systems ruled the roost in terms of production systems, and still do in some cases. DMSs are still unbeatable for high volume production, or if tact time is very short. The FMS which followed the DMSs focused on the flexibility of the system. The objective was to cover a number of different applications with the same Hardware (HW). However, the adaptability of such systems was never good enough, so these systems were succeeded by RMS [21]. In RMS, the system is composed of modules which can be reconfigured for a specific purpose. If a system is to be reconfigurable, then that presupposes a certain degree of modularity. RMS is better suited to perform specific tasks than FMS, and it neglects the costs and compromises needed for flexibility in case of FMS. Recently,

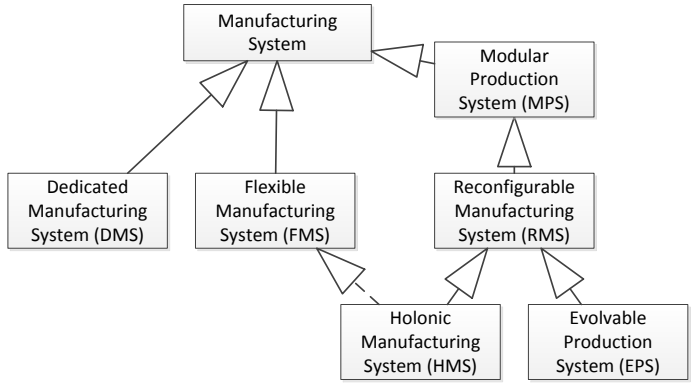


Figure 3.8: Relations between selected production paradigms

newer production paradigms have come along, in the form of HMS [137, 139] and EPS [98, 118]. These are better at including the process and control perspectives, and offer a more comprehensive approach to production system design and implementation.

The relations between the paradigms referred to here are illustrated in Figure 3.8. This uses notation of a UML class diagram, and it illustrates how the various paradigms inherit from each other. The figure shows the historical perspective, as time evolves from left to right and top-down. The figure highlights the basic prerequisites for RMS and its successors, which is that the system must first have modularity before it can enable reconfiguration. Thus, the Modular Production System (MPS) precedes it.

As mentioned before, the terms Production, Manufacturing, and Assembly System are used interchangeably in the context of this thesis. The *[#] Manufacturing System* is used in this section, as it is the dominant term referring to such systems in the literature.

3.3.1 Dedicated Manufacturing Systems

[84] defines Dedicated Manufacturing System (DMS) as follows: "A machining system designed for production of a specific part, and which uses transfer line technology with fixed tooling and automation." DMSs are traditional manufacturing systems, which are designed and built for dedicated purpose only from inexpensive fixed automation components. DMSs are most efficient for high volume and low mixture type of production from both the performance and cost point of view. DMSs are normally used for producing a single product that is often the core part or product of the company. DMSs generally work optimally for high production rates, for long run-times, and for producing output that is sufficient in quality. According to [72], DMSs are designed for limited ranges of production requirements. Their performance is high and inherently robust, but they do not provide the required responsiveness to change. DMSs are the least expensive solution in terms of initial capital cost, especially for the first produced product type. A DMS may be designed to produce closely-related variants of the same part. In such cases, all the variants must be known before the system is designed. Such systems are hard or even impossible to re-use after the end of their life-cycle, because they are so tightly dedicated to the manufactured product.

DMSs are characterised by the considerable engineering effort that goes into system design, system build-up, and the ramp-up phases. Usually they are one-of-a-kind systems, so each delivered system is a kind of prototype. Companies do of course try to utilise their previous knowledge and production system designs as much as they can to increase the reliability, robustness and performance of the system, and most of all, they try to get the system right at the first try.

However, this is often hard to do at the design stage, because each production system is unique. Attempts have been made to reduce costs and delivery times through the use of modularisation and re-usable Commercially-available Off-The-Shelf (COTS) components. However, this is extremely difficult to achieve at higher system design levels, especially if the systems are not similar.

DMSs are good in their own way, though. Koren [69] states that if a DMS runs at a minimum of 75% of the line's maximum capacity, it is the most economical solution. Also, in cases when the performance requirements are very tight, DMSs are unbeatable. However, because of global competition and fluctuating market demand, there are many situations in which DMS operate far below their full capacity (e.g. average utilisation is only 53% [69]), which means that the producer is probably losing money. The DMSs are not capable of coping with the challenges of severe fluctuations in product demand, because they lack flexibility, adaptability, reconfigurability, and scalability.

3.3.2 Flexible Manufacturing Systems

ElMaraghy [21] defines a Flexible Manufacturing System (FMS) as follows, "It is an integrated system of machine modules and material handling equipment under computer control for the automatic random processing of palletised parts." FMS are systems that are intended to be able to produce a variety of products with the same production system. They normally contain programmable machines, like the Computer Numeric Controller (CNC), which can be adapted to produce different parts merely by changing the recipe program, the product fixtures and the tools. These changes can take place automatically through the use of the program memory, the pallet exchanger, the storage system, and the tool magazine respectively. In contrast to DMS, the FMSs are not designed around a specific part or product, rather, they are general-purpose programmable machines. They are designed and built around a standard operational envelope and purpose, regardless of, and well before, the final user determines what they want to manufacture or build with the machine. As the makers of these machine do not know the final intended use beforehand, the machines and components for an FMS need to be designed so that all possible functionalities are built-in. One of the problems with this, though, is that the system's full functionality is often under-utilised, which implies a waste of investment capital. [69]

The objective of an FMS is to produce, cost-effectively and at the required quantity and quality, several types of parts from a part-family with the same system. The part-family can be changed over time with minimum changeover costs. [21] The objective of an FMS is defined in [72] as its being designed for a broad range of production requirements. Even though they are inherently responsive to change, they do have drawbacks in that they are often more complex than necessary, their performance is less robust than a DMS, and they are simply too expensive for many applications.

FMSs are suitable for manufacturing: a) any kind of product fitting into the given work envelope and processes, b) any mix of the parts, and c) in any sequence. If only small quantities of parts are needed, then the parallel-type FMS is probably the most economical solution. Compared to a DMS, the production capacity of an FMS is usually lower, while the initial cost is much higher. [69]

FMSs do have their strengths. They are easily scalable in capacity, flexible and convertible, and can thus produce small quantities of different parts economically. However, the advantages of the FMSs are often outweighed by the disadvantages, which are, for example, [69]: lag of performance as single tool operation; production capacity is lower than DMS; because they need to be fit for a variety of products and processes they need to carry a number of extra capabilities which are not usually utilised; and most of all, the high initial cost of the machines and the system.

3.3.3 Reconfigurable Manufacturing Systems

The concept of the Reconfigurable Manufacturing System (RMS) was first introduced by the Engineering Research Center of the University of Michigan in the mid-1990s. Koren [69] defines an RMS as follows: "An RMS is a system designed at the outset for rapid change in structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family." In the context of RMS, a *part family* is defined as all parts (or products) that have similar geometric features and shapes, the same level of tolerances, require the same processes, and are within the same range of cost [69]. From a broader perspective, this would mean, e.g. several types of microprocessors or several types of engine blocks.

The objective of an RMS is to provide exactly the functionality and capacity that is needed by the system, and to do so exactly when needed. [21, 69] The objective is defined in [72] that RMSs are designed to possess customised flexibility, so that they are designed for specific ranges of production requirements (as opposed to a single set of production requirements in the case of a DMS or a wide range of production requirements in the case of an FMS) and they are even customised to the initial production requirements. Both the hardware and software in an RMS can be reconfigured cost-effectively when production requirements change, or new ones appear.

According to Koren [69] the RMS shows reconfigurability on two levels: 1) in the arrangement and connections of the machines at system level, and 2) at the machine level, in that a machine itself is reconfigurable. At both these levels, the systems are designed according to specific design principles and they possess special characteristics, which are the core characteristics of any RMS [69]: modularity, integrability, customisation, scalability, convertibility and diagnosticability. These characteristics are the foundation stones of this work and are fundamental requirements for the proposed production description model and the module descriptions. Therefore, they are defined here explicitly and discussed in more detail below.

1. **Modularity:** All the main components of an RMS are expected to be modular. If needed, any modular component may be replaced and upgraded with another to better suit a new or changed application. The modularity makes the machine easier to maintain and update than would be possible with one monolithic machine such as might be used in a DMS. This lowers the life-cycle cost of the system as its useful working life can easily be extended. The selection of the basic modules for a production system, and their connection methods, inevitably leads to the creation of systems that are easier to integrate, diagnose, customise, and convert than they are in a DMS. [69, 72]
2. **Integrability:** The modules must be smoothly and easily integrated together. In order to do so, cross-company integration methods need to be available. This can be achieved by, for example, defining a set of system configuration and integration rules for the modules for a target sector. This requirement should be taken into account at all levels of system and module design, including the mechanics, controls, communications, and energy supply. [69]
3. **Customisation:** This is the main distinguishing characteristic between an RMS and either a DMS or an FMS. The RMS reduces both system and machine costs by combining the productivity of the DMS through the use of multiple tools, and the flexibility of an FMS in being able to handle different part variants within the selected part family. It enables a production system to be designed for a specific part family, which will ensure that most of the manufacturing resources are utilised for the production of every member part. Customisation means that the dominant features of the part family will determine the overall system and machine configuration. [69]
4. **Scalability:** Scalability means that the number of modules in a machine or system can be increased or decreased as needed. The modules and processes can be multiplied or removed from the system. Not only modules, but complete machines can be added to or removed

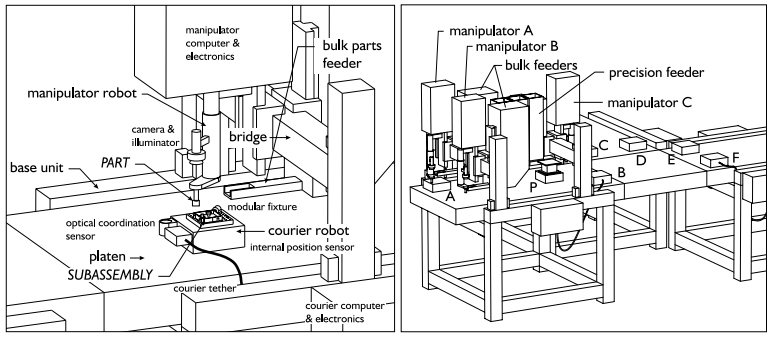
from the system. Thus, the system's capacity can easily be adjusted to respond to demand. Koren sees scalability as the counterpart to convertibility [69].

5. **Convertibility:** Convertibility appears at several levels in an RMS. At the machine level, conversion takes place when changing over production between two parts belonging to the same part family, or, for example, by manual adjustment such as in passive Degrees of Freedom (DOF). The changeover must be carried out quickly in order to keep the system up and running. The changeover time can be related to the frequency of the change, e.g. in the case of a daily change, a 1 to 10 minute changeover time would be acceptable. To make this possible, the RMS must possess advanced mechanisms for easy conversion between system configurations. This includes advanced sensing and control methods that helps in the speedy recalibration of the machines after conversion. At a higher level, convertibility may include adding completely new functions to the machine, or at system level, adding new machines to expand the functionality of the system so that it is able to produce new parts. [69]
6. **Diagnosticability:** According to Koren [69] diagnosticability has two aspects: 1) detection of machine failure, and 2) identification of the causes for unacceptable part quality. The latter is regarded as critical for an RMS, because when the production systems are made more reconfigurable, and layout changes take place more frequently, the ratio of time to quality becomes essential. It means that the recently reconfigured system must be adjusted and tuned quickly in order to produce parts of sufficiently high quality. [69]

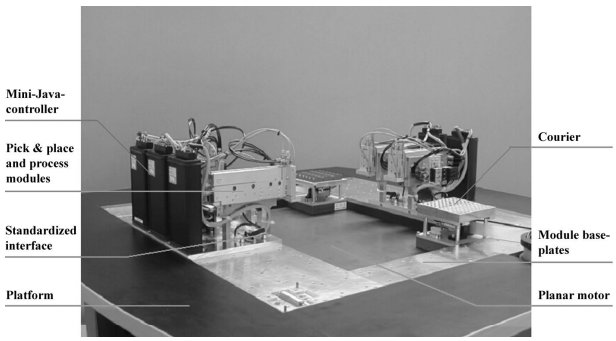
The RMS's rationale is to utilise dedicated, flexible, and reconfigurable manufacturing equipment. The decision regarding which type of manufacturing system to utilise in an RMS factory is strongly dependent on the production volume, production requirements, and the expected frequency of change in those requirements. Similarly, the decision regarding what type of manufacturing equipment is utilised at each station is strongly dependent on the operational requirements. [72] If larger quantities are needed and the market uncertainty is high, then an RMS is the most cost-effective solution [69].

Moon [89] has identified the justification for the production system design concept presented in this thesis. His paper focuses on the system design of an RMS and how to determine all the feasible configurations for a reconfigurable machine. Moon defines the overall methodology, covering the whole process chain from the initial design phase of an RMS to the deployment of an operating tool. His method has various phases including requirements and functions analysis, mapping of functions, and the selection of modules for the feasible configurations. Finally, the configurations are evaluated from several perspectives, according which the optimum configuration can be selected. The paper points out that the key to the proposed approach is to have available a unified (standardized) way of modelling the machining requirements, the machine modules and the machine tools. Moon's paper clearly indicates the pressing need for standardisation and data sharing. [89] This thesis aims to provide a unified data-sharing model as defined by Moon, specifically for the machine modules. These are collected as a library or libraries, which can be utilised during the module selection stage of production system design as they fulfill the requirements for data sharing and standardisation.

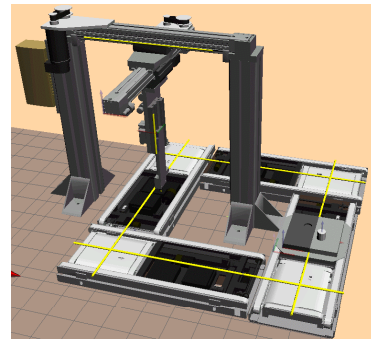
Figure 3.9 shows examples of research platforms for RMS in the domain of assembly. The Agile Assembly Architecture (Figure 3.9a), is a system for microfactories which has been presented in Hollis and Quaid [46] and in Hollis and Gowdy [45]. Gaugel, Bengel, and Malthan [40] presented the MiniProd (Figure 3.9b), which is a construction-kit type of assembly system built from ready-made modules. It is intended for lab automation, and is characterised by a planar motor, which takes the processed entities, such as assembled products, to the processing stations around the planar plate. Later, Festo took up the concept and increased its technological readiness level towards the level required in industrial applications [101]. Martinez Lastra [83] presents



(a) Agile Assembly Architecture [45]



(b) MiniProd [40]



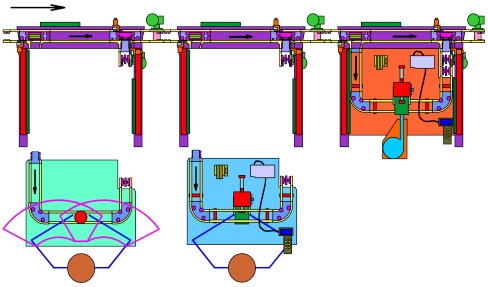
(c) ABAS [83]

Figure 3.9: Examples of research platforms for Reconfigurable Manufacturing Systems

another approach to building modular and reconfigurable assembly systems with the Actor-based Assembly Systems (ABAS) (Figure 3.9c).

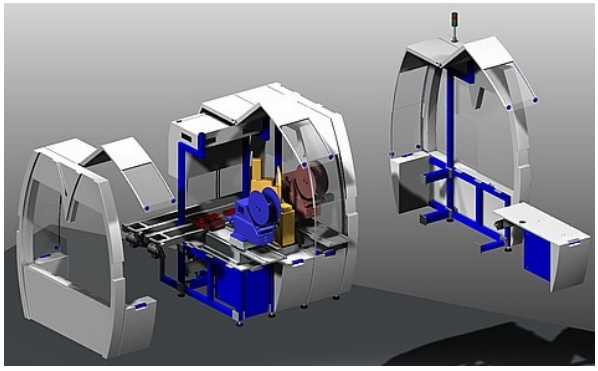
There are not many implementations of RMS in industry. Many systems are basing on modularity, but characterising a system to be an RMS or an FMS, is not a straightforward task. There is limited publicly available information about the production machines and processes, and much of this is intended for marketing purposes. However, driven by its own development needs, CNC machine tools were the first area in which the RMS concept was seriously taken up. One of the first practical prototypes was Arch-Type RMT [69].

Figure 3.10 shows a few commercial examples of RMS in the domain of assembly automation. Figure 3.10a illustrates the Flexline from SMH Automation, which is used for semi-automatic and automatic precision assembly. It demonstrates reconfiguration in the form of exchangeable process tables, which are instrumented according to the specific process needs. The table frame offers the embodiment of a cell, transportation for product pallets, and the distribution of services. Figure 3.10c shows the Modutec platform from IMA Automation [150]. This modular assembly system is composed of a set comprising framework, logistic and process modules, which are changeable. The cells provide logistics for the pallets carrying products. Figure 3.10d shows another example of reconfiguration with an exchangeable feeder / processing station for a rotary indexing cell, which the operator is currently attaching to the system. Figure 3.10e illustrates the EcoLine platform from Mikron [151]. This platform is composed of standardised modules comprising: electrical cabinet, logistics module, process module(s) and handling unit(s). The process module contains the cell frame, and it is thus quite similar to the Modutec platform. The

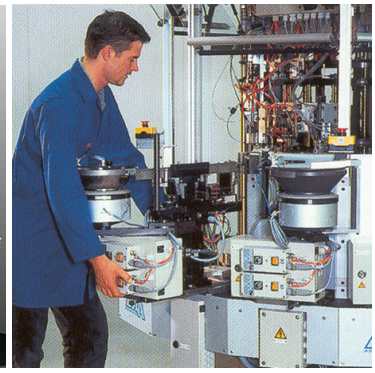


(a) SMH Automation. Flexline.

(b) ASM. Siplace SMT placement line [149]

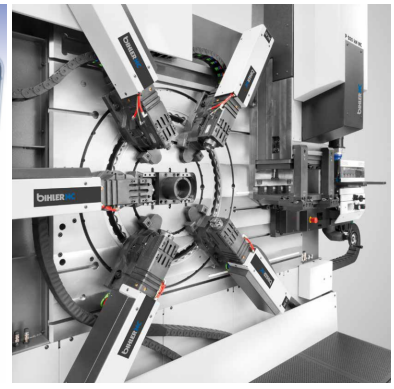


(c) IMA Automation. Modutec [150]



(d) IMA Automation. Rotary indexing system. [150]

(e) Mikron. EcoLine [151]



(f) BihlerNC. [152]

Figure 3.10: Examples of commercially available Reconfigurable Manufacturing Systems

process modules can be combined together in multiple configurations. Likewise, the handling units can also be inserted into the process modules in multiple positions and configurations.

Figure 3.10f shows a forming and stamping machine from Bihler [152] with a reconfigurable Numerical Controller (NC). This is really getting to the essence of a successful RMS. The current set of tools can be automatically reconfigured into new configurations, as can the number of tools and processes used. Bihler already have reconfigurable assembly systems in their portfolio. Their platform builds on a base mounting block that has matrix patterned fastening holes. The fastening area of a base can be extended with separate parts. A wide variety of process and peripheral units can be mounted on the base in a wide variety of configurations. Their processes include pick & place, press, welding, screw insertion, and material feeding. These processes are all integrated under a centralised control system.

Figure 3.10b shows a Surface Mount Technology (SMT) placement machine from ASM. This is to give a broad view of the SMT manufacturing process, which displays many of the features of RMS, especially with regard to the placement machines. This business sector has a heterogeneous landscape in which the various competing players have specialised their operations for different sections of the overall SMT manufacturing process. These sections, such as component supply, placement, soldering, re-flow, and inspection, all need to be brought together in a single, integrated system, even though the parts may come from different sources. This is impossible without standardisation of both interfaces and processes, which are the enablers for integration and reconfiguration. An example of intra-cell level reconfiguration in SMT manufacturing can be seen in material supply. The components installed in the end product are supplied to the production system with different tape reels, tubes, and trays. As these follow agreed interface standards, they can be fitted into different feeders. The feeders have a mounting interface to the placement machines. These feeders can be arranged in different configurations and filled with different components according to need. In a few cases, the feeder interface of a dominant player in the business has become established as the "de facto" standard, such as Fuji or Panasonic, whom the other suppliers have started to follow. Thus, exchangeability and interoperability between process modules is beginning to happen.

Among automation components, reconfiguration is well-represented, even at the element level. There are examples of reconfiguration in control automation (modular Programmable Logic Controller (PLC) and Input and Output (IO) systems), and pneumatics (cylinders, their mountings, and auxiliary devices). In these cases, the companies have made extensive module platforms for their own product families, in order to be able to answer the varying needs of their customers. Variability is provided through component interconnectivity and a large variety of combinations, achieved through modularization and the standardization of the internal interfaces and components. With such an arrangement, they have been able to reduce the total amount of items in the portfolio, gaining advantages in economies of scale and inventory management.

Apart from the SMT case, which does demonstrate some common interfacing and cross-field exchangeability, one of the major issues with all the above-mentioned examples is that the module platform and interfacing is company specific, which limits any reconfiguration to the modules from single supplier. It is clear that the next step towards realizing the RMS concept in a cross-vendor environment is shared (reference) architectures and the standardization of interfaces and processes using common information models and shared resource data.

3.3.4 Holonic Manufacturing Systems

Van Brussel [137] laid down the foundations of Holonic Manufacturing System (HMS). An HMS exhibits holonic behaviour, which means that the system is composed of several subsystems, each of which can behave in a completely autonomous way. These subsystems (try to) combine their

efforts to achieve an overall system goal. [137] Such systems are composed of holarchies and holons, which are defined as follows:

a) The holon: An autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. A holon can be part of another holon. The holons are simultaneously both a part and the whole. [137, 139]
b) The holarchy: A system of holons that can co-operate to achieve a goal or objective. The holarchy defines the basic rules for the co-operation of the holons and thereby limits their autonomy. [139]
c) The HMS: A holarchy that integrates the entire range of manufacturing activities from order booking through design, production, and marketing to realise the agile manufacturing enterprise. [11, 138]

[139] proposes a reference architecture for HMS composed of three types of basic holons: order holons, product holons, and resource holons. These are supported by additional staff holons. The descriptions produced for this thesis can easily be used to describe the data needed for resource holons.

HMSs are strongly related to control systems in the domain of multi-agent systems and, more precisely, to autonomous cooperative agents for distributed manufacturing [138]. Reconfiguration from the controls viewpoint is the main focus of this work. Therefore the underlying physical system can be inherited from either RMS or FMS paradigms, but the overlying control system follows the HMS paradigm. This inheritance from two parents is illustrated in Figure 3.8.

3.3.5 Evolvable Production Systems

The Evolvable Assembly System (EAS) is a recent paradigm for assembly and production systems [98], and is regarded as an improved version of the RMS, (the paradigm name was later changed to Evolvable Production System (EPS)). Borches and Bonnema [10] have given us a detailed review of evolvability. In their review they define the key drivers, and produce a clear definition of evolvability which they discuss in the context of Design for Evolvability, i.e. how evolvability can be designed into a complex system.

Overall Description

Onori [98] provided the first general definition of an Evolvable Production System (EPS) (back when it was still known as an EAS). He highlights the problems with the FMS, which the new EPS paradigm was intended to overcome. According to Onori [98] the main issues with FMS are: a) Its flexibility is not optimum for performance, but it offers an expensive, highly technological system that is only fairly adequate for different product types; b) The strong hidden relationship of product and production system, where planned products need to be manufactured by existing systems. This often ends up in a situation where both the new products and the new production equipment need to fit with the existing production environment. Thus, focusing on product design and the production system as separate entities will eventually fail, and c) the main issue, which is the FMSs' ability to deal with uncertainty. New systems need to be able to adapt to new products (including types and families) and changing market requirements (volume, lifespan, etc.) with ease. Therefore, EPS was proposed to overcome these issues. It would be based on a number of re-configurable, task specific modules that would allow continuous evolution of the production system. Simultaneously, the dynamic link from product design to production processes is maintained. [98]

Onori [98] summarises the essence of the paradigm change from FMS to EPS as follows: The theoretically very flexible, multi-purpose cells of FMS are replaced by a highly flexible concept consisting of several well-targeted but not, intrinsically, highly flexible components. Another key aspect is that the product design procedures must be an integral part of the production system development phase, and EPS follows the idea of evolution rather than adaptation. In the two latter points, in particular, this is the main area of its divergence from RMS.

Semere et al. [118] state that the main objective of EPS is system adaptability, which it aims to achieve through the following four aspects [118]:

1. *Optimised functionality*. The assembly equipment is kept as simple as possible by deriving small, dedicated, and process-oriented modules. These may be then interconnected to form production cells, lines and systems.
2. *Optimised orchestration*. The control system needs to fulfil the most agile aspects, as the more complex layers above (device, module, cell, and system) start to bind the flexibility and degrees of freedom. High agility is achieved by adopting a distributed control approach with embedded controllers (e.g. a multi-agent based solution).
3. *Adaptability and Evolution*. The modularity allows stepwise upgrading and economic flexibility (it is cheaper and simpler to change a module than modify a system). Interface standards aid in connectivity and exchangeability. The actual system may also adapt to minor changes via its control system, which, being skill-based, allows for emergent behaviour to be exploited.
4. *Robustness*. The equipment is dedicated, small, and includes its own processor. Other modules, such as robots, may even be reconfigurable themselves. The control system is goal-oriented, and the system is process-oriented. This results in a dedicated system based on an adaptable concept with advanced interfaces. Robustness is increased by the use of existing and tested production modules, which are thus expected to operate more reliably, even though they are not directly-connected entities.

Maraldo et al. [80], Onori, Barata, and Frei [99], and Semere et al. [118] continued formalising the basics of the EPS concept. Maffei [78, pp.19–25] puts together an introduction to, and the essential characteristics of, an EPS. He concludes with a table illustrating different manufacturing systems, namely, DMS, FMS, and EPS. Onori et al. [101] present developments for taking the EAS concept to next Technology Readiness Level (TRL) towards industrial implementation.

Definition and Characteristics

Having established that the EPS is a production system design paradigm that is a successor and derivative of RMS, it can now be characterised by the following definitions:

a) It proposes a highly flexible concept consisting of several well-targeted but not, intrinsically highly flexible components. These components are pre-designed, tested, and available from catalogues.
b) Another key aspect is that the product design procedures must be an integral part of the production system development phase. I.e. there are direct and strong links from the product requirements to the production system (modules and layout) through the capability definitions. This enables system creation through the reconfiguration of existing modules.
c) The above point creates the foundation for the true re-use of modules, making it both possible and affordable to implement changes in the production easily.
d) EPS follows the idea of evolution rather than adaptation.

EPS aims to achieve adaptability through: a) breaking all dependencies between existing production system(s) and product design, thus allowing innovative product design to take place; b) the product design is then followed by independent process selection procedure, resulting in an optimal assembly system principle, and c) the optimal layout is then linked and built up from a

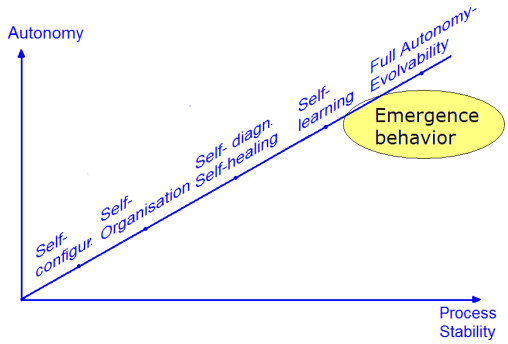


Figure 3.11: Evolution of Manufacturing systems. Relation Autonomy-Process stability. [79]

broad range of finely granulated process-oriented components with standard interfaces using an EAS methodology. [98]

EPS exhibits the following characteristics [118]:

- it is a fully reconfigurable system platform that exhibits an emergent behaviour
- it is composed from mechatronically integratable assembly units
- the reconfigurable system has to be composed of process-oriented components, which provide granularity for emergent behaviour
- a system that can automatically determine its functionality based on the components' skills
- the change in paradigm is, partly, that we no longer invest in the programming and coding, but, rather, in how to establish and exploit relationships
- maintenance, documentation, and the ability to store information support the operational stability of the system

Frei [37] and Frei, Ferreira, and Barata [38] discuss the self-organisation aspects of EAS, while Maffei et al. [79] discuss its self-configuration and autonomy aspects. The evolutionary aspects of a manufacturing system, especially with regard to the EPS, are illustrated in Figure 3.11. The autonomy can be built into the system only in a sequence of ordered steps. The first step must be completed, before any approach can be made to the next step. The EPS aims to promote emergent behaviour, so there is a set of fundamental technological steps which must be taken first. These steps have been identified as self-configuration, self-organisation, self-diagnostics and self-learning. Only if these characteristics have been ensured can the technological readiness for evolvability and emergent behaviour be said to have been achieved.

3.3.6 Summary of Production Systems

Of the production systems presented above, DMS, FMS and RMS, can be regarded as the three basic concepts behind current production systems. RMS can be further divided into different, related, and more developed systems such as Agile, HMS, or EPS. These are like derivatives of RMS with a different emphasis, i.e. the differences between them are not that great. In fact, FMS, RMS, HMS, and EPS, all define environments in which the concept behind the descriptions produced for this work could well be utilized.They also pose the requirements which need to be taken into account and fulfilled by the concept and resource descriptions. Table 3.1 summarises the various features from different manufacturing systems. The columns in the table are sorted in chronological order of appearance.

Table 3.1: Features of different manufacturing systems. Elaborated from [21, 69, 70].

	DMS	FMS/ CNC	RMS	HMS	EPS
System Focus	Part	Machine	Part family	Process	Product and Production process
System Structure	Fixed	Adjustable	Adjustable	Adjustable	Adjustable
Machine Focus	HW	HW & Ctrl	HW	Ctrl	HW & Ctrl
Machine Structure	Fixed	Fixed	Adjustable	Adjustable	Adjustable
Flexibility	No	General	Customised	Customised	Customised
Scalability	No	Yes	Yes	Yes	Yes
Productivity	High	Low	High	Mid / High	High
Simultaneous Operating Tool	Yes	No	Yes	Yes	Yes
Reaction to Change	None	Possible/ Automatic	Possible	Automatic	Automatic
Learning Capabilities	None	None	None	Yes	Yes
Self Configuration	None	None	None	Yes	Yes
Self Organisation	None	None	None	Limited	Limited
Self Recovery	None	None	None	Yes	Yes
Lifetime Cost	Low	High	Intermediate	Intermediate	Intermediate

3.4 Description Languages

Various kinds of description languages exist in the field. In this literature review, the focus is placed on two main areas - system descriptions and module descriptions, which are specifically targeted at the field of production automation or factory automation. These can provide or capture system design features or provide a description of a specific aspect of the system. Our main interest is in solutions that try to capture thoroughly the module information, which is required for system design and commissioning and, can be provided publicly by the module provider. These descriptions create the competing, or complementing, solution space for the proposed concept.

3.4.1 System Description

The system descriptions define the structure and characteristics of entire production system(s). They capture the information about the components composing the system and how these components are connected to each other. Of specific interest in this context are the descriptions that contain partials or sections, which can be used to express fundamental production modules, and can be created and distributed by the module provider.

3.4.1.1 Automation ML

The perspective of AutomationML is to standardise data exchange in the engineering process of production systems i.e. from the planning of production engineering to commissioning and ongoing operation of equipment, thus bridging the gap between production and automation planning. AutomationML defines a concept and a neutral data exchange format, which is intended to be lossless and scalable. It is aimed to reduce conflicts and to improve the interoperability and interconnectivity between a number of heterogeneous tools used in various phases and disciplines of automation system design and engineering. These include disciplines like mechanical plant

engineering, electrical design, Human Machine Interface (HMI) development, PLC and robot control. AutomationML [2] states that around 60% of the costs of an automation system come from the engineering, and a great part of this is re-doing and re-defining the same things to different tools, instead of re-use. The typical engineering value creation chain includes a number of sub-suppliers or departments, which increases the number of different tools in use. This all comes to a head in the form of data exchange issues, which can create significant bottlenecks in industry [19]. A lot of wasted time and effort can be saved by increased interoperability and more efficient and thorough data exchange between the tools for a production system design. At the same time, the quality of the data and the system design are expected to improve as well, as the focus can be placed on the right things. [2, 4, 19]

The AutomationML intended to thoroughly support the incremental design process. The system design can actually be started from incomplete sets of information, such as factory layouts, templates, and rough sequences. These are incrementally developed, step by step, during the engineering phases. The AutomationML should retain all the information gathered throughout the design process and offer a medium for information exchange between tools, from the basic tool descriptions up to complete control system and it should not lose any piece of information in between. [2]

AutomationML is driven and created by seven companies and three universities, mainly from Germany. The work on the project started around 2006 and the first drafts for an *intermediate format for the Digital Factory* were published in 2008. An independent registered association was founded in April 2009. In 2013, three new collaborations were started with eCl@ss, OPC Foundation, and ProSTEP iViP. The first is focused on the exact classification and description of materials and products, and offers a standard for unambiguous information exchange. The OPC Foundation offers the use of Open connectivity via open standards (OPC) technology for descriptions, and the third, ProSTEP iViP, is creating a link to the Standard for the Exchange of Product model data (STEP) descriptions. Currently (in July 2014) the association has 18 industrial members and 11 academic members. The current AutomationML specification version 2.2 is dated to July 2013 [3] and contains four parts. There is an ongoing process to publish all these parts as international standards under the auspices of the International Electrotechnical Commission (IEC) and the first part was published as IEC 62714-1 at 26 June 2014 [53]. [2, 4, 19]

The AutomationML association develops and maintains an open, neutral, eXtensible Markup Language (XML) based, and free-to-industry data representation standard which enables a transfer of engineering data which spans individual companies in a whole domain. Their strategy is to utilise already-existing formats by use, adaptation and extension or merge. Instead of developing completely new data formats for the intended purpose, they are combining and aggregating existing ones in applicable ways. The representation of plant-specific data includes general sections and specific data about the plant's structure, geometry and kinematics, and logic descriptions. The association is planning to append later the representations of networks, mechatronical systems, and others. Hence they state that AutomationML is the most comprehensive data format for plant engineering. It is already used in the field and is also available in several commercial products through import and export features. [4]

AutomationML describes real plant components as objects encapsulating different aspects. An object can consist of other sub-objects, and can itself be part of a bigger composition. It can describe a screw, a claw, a gripper, a robot or a complete manufacturing cell or line in different levels of detail. The content stored by the AutomationML file contains four main facets i.e. topology, geometry, kinematics, and logic. The references and relations between entities are also included. The topology contains the relationships between objects presented in a hierarchical structure, and the properties of these objects. The geometry represents the 3D information and all graphical attributes of the model. This is mainly done by referencing an external COLLAborative

Design Activity (COLLADA) object, which is intended to represent the visual model of the system. The kinematics represents the kinematic model and kinematic chains of the system by modelling the mechanical connections and dependencies between the objects. The main intention is to support the motion planning tasks. The kinematics part is also included as reference to an external COLLADA object. The logic section focuses on modelling the behaviour and sequence of actions, the internal behaviour of objects, the controls and the IO connections between the objects. In the case of logic description, part of the information, such as the IO connections, is stored directly into the AutomationML file, while the logical behaviour itself is stored in an external file in the format of PLCopen XML (PLCopen XML) (See Ch. 3.4.4). [2, 4]

The AutomationML strategy is to re-use other standards, instead of developing new ones. The re-used parts of AutomationML are:

- CAEX (IEC 62424) as a top-level format and for plant topology (See Ch. 3.4.1.1),
- COLLADA for geometry, kinematics and motion planning,
- PLCopen XML (from PLCopen - organisation (PLCopen)) Sequential Function Charts (SFCs) for behaviour and sequence descriptions (See Ch. 3.4.4),
- MathML for mathematical formulas,
- Future: eCl@ss for component or material classification,
- Future: OPC Foundation for OPC data items, and
- Future: STEP file integration

SmartComponents Bartelt, Schyja, and Kuhlenkötter [7] have proposed an extension to AutomationML in the form of digital representations of real objects, called SmartComponents. These are represented as an AutomationML file with additional semantics. They provide a framework which centrally (there is a server application and Application Program Interface (API) for it) takes care of changes and partial representation or modification of the project files. They state that this approach prevents the possibility of data losses, especially in cases where the application only partially understands and utilises the project file. The main idea of SmartComponents is to create a model of a component or device, such as a robot, gripper, or the like. This model is then stored in a library, from where it can be distributed and utilised by others. The system designer can compose the target application by collecting such components, and linking and relating them together as a working application. [7, 117]

The SmartComponents contain features like kinematic structure, 3D model, interfaces, and also logical behaviour, which then can be shared by, e.g. various simulation tools. All this is inherited from its basis in AutomationML. This aims to facilitate the modelling only needing to be done once, after which it can be utilised by any other application. Non-standard, black-box representations of actions (e.g. gripper movement actions) can be expressed through the API. [7, 117]

The seeming advantage of having a central database, and especially accessing all information only through an API for storing all project files might turn out to be the project's main weakness and limitation. Another issue might arise from the central hosting policy. This will only work if the server application can be easily reproduced with suitable licensing by the project data owners rather than being dependent on an external service provider.

Summary AutomationML presents the topology, the 3D geometry and kinematics, and the control data. Although it works well as a representation of the exchange format for the design of a plant or manufacturing system, it doesn't work very well as a data source and distribution format for the production modules before the system design starts. Therefore, the proposed concept offers an exchangeable, lower level source of information about production modules that can be transferred to AutomationML, so the proposal can complement AutomationML. Many of

the concepts behind AutomationML are in line with the concept behind this thesis. Actually, AutomationML could fill missing gaps in our proposed design framework, especially with regard to, e.g. document 3 (and part of document 4) in Figure 5.10 (p.98).

What AutomationML lacks is the abstractions in the early phases of the production system design. It jumps directly to existing modules through the concept of templates. It does not provide an abstraction of the manufacturing process design, and getting from there to the capabilities of the devices, although this link does exist in the format. The harmonisation of manufacturing processes is not the objective of the standard. It is something that is left for the user to decide, case by case. The abstracted system design phase is also missing at the beginning of the system design. The system is not designed independently of the selected and implemented module(s) in the first phases of the design. It does not offer the user the possibility to design the system at an abstract level, and then to evaluate alternative options and technological solutions performing the same operations without locking the design into a particular implementation at this early stage.

The basic assumption of AutomationML is that the templates of the modules are inserted into the system description, but they are always customised and modified extensively for the project's purposes. This means that the modules are not stable and re-usable entities, but are engineered or customised over and over again. Also, the concept requires that the internal design and implementation of a module is revealed extensively and completely for the system design. Of course, the generality of the concept gives it weight. It is more probable that the same concept can cover all unexpected situations that could arise in a realistic production system design landscape.

3.4.1.2 Systems Modelling Language (SysML)

Systems Modelling Language (SysML) is a requirement-driven, general purpose modelling language used for system engineering, and it has close relationship to UML, sharing a subset with it. The publisher of the specification, Object Management Group (OMG), defines it as follows: "SysML is a general-purpose graphical modelling language for specifying, analysing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modelling system requirements, behaviour, structure, and parametrics, which is used to integrate with other engineering analysis models." [153]

A few studies have looked at utilising model-driven design for control system creation, and to create a direct link from SysML to PLC controller languages, especially to IEC 61131-3 [52] [13, 64, 134] and to IEC 61499 [50] [135]. A complete Software (SW) tool chain for making such a transformation is discussed in [105].

SysML intends mostly to express and extend the UML kind of system engineering information, to exchange this information between tools, and to show and edit this information for humans. It mostly targets system level information, like AutomationML. The concept of *Block* and *Ports and Flows* presented by [97] might be useful for connecting modules together and defining the interfaces. Its main drawback is its generality. It is capable of describing almost any kind of system, but its semantics and formalism are too weak for, say, a production environment. Additional templates are required if the content is to be harmonized, as well.

3.4.1.3 Other descriptions

MANDATE The objective of MANDATE (ISO 15531) is to aid effective manufacturing resource planning in operational, financial, and simulation domains by standardising the data models (semantics) and data exchange. It can express processes and information or material flows. It is scattered over several smaller partial standards, defining specific cornerstones for data exchange

of Enterprise Resource Planning (ERP) and Manufacturing Execution System (MES). One of the parts, [58], focuses on the representation of the manufacturing resources and their activities, operations, and functions (i.e. the capacities and capabilities). It specifies the information model needed to define, operate, and monitor a resource. [17, 18]

However, it targets higher-level planning aims, and it only defines the semantics and granularity on a very coarse level [17]. Therefore, it requires a lot of linking to external resources and definitions in order to define, e.g. the characteristics and properties of a resource, which are not thus formalised by the standard. Its applicability for defining production resources for design and selection purposes remains weak, as that is beyond the scope of this standard, which focuses mainly on SW integration in manufacturing operations.

3.4.2 Module or Device Description

The module descriptions define the properties and features of a single module. Normally such description is then shared and distributed over various channels, and later utilised by a set of tools used for design, commission, or operation in a production environment. Many such descriptions are linked to all of these phases, and in many cases they are looked at from the control and automation point of view.

3.4.2.1 Unified Manufacturing Resource Model (UMRM)

Unified Manufacturing Resource Model (UMRM) is a modelling method for CNC machine tools and their auxiliary devices, such as tools, tool holders, fixtures, pallets, conveyors and handling robots. It is a unified model to exchange machine-tool related resource information between Computer Aided Process Planning (CAPP) tools. It utilises STEP and STEP-NC, and presents information in a compatible form. [141, 142]

UMRM is capable of representing the resource units as elementary elements, which can be assembled into larger entities using the same unified model. Finally, it can be used to describe the entire system configuration of a CNC machine tool. Thus, it represents not only the pure resource units, but the systems as well. The UMRM uses *mechanical machine element* as elementary building blocks, which can contain other such elements. This arrangement can flexibly represent any resource model, even customised machine components. Each element is fixed to a specific kind of functional machine tool element, selected from predefined lists, which at the same time define the structure and parameter set for the described element. This all binds the model tightly to the domain of machine tools. [141, 142]

The domain specialism is highly present throughout the UMRM, as it focuses only on the domain of machine tools. The model contains strong typing and categorisation of information by using the elements and terms from the domain field. Thus, it can be regarded as a domain-specific language whose generalisability to other domains remains questionable.

3.4.2.2 Open systems application integration framework ISO 15745

The ISO 15745 *Industrial automation systems and integration - Open systems application integration framework* has five parts: Part 1, *Generic reference description*; Part 2, *Reference description for ISO 11898-based control systems* (i.e. Controller area network (CAN) based networks); Part 3, *Reference description for IEC 61158-based control systems* (i.e. industrial fieldbus based networks); Part 4, *Reference description for Ethernet-based control systems*; and Part 5, *Reference description for High-Level Data Link Control (HDLC)-based control systems* (ISO 13239).

The introduction to the standard (ISO 15745-1:2003) states that "Application systems are developed from application specifications that typically contain textual descriptions, diagrams,

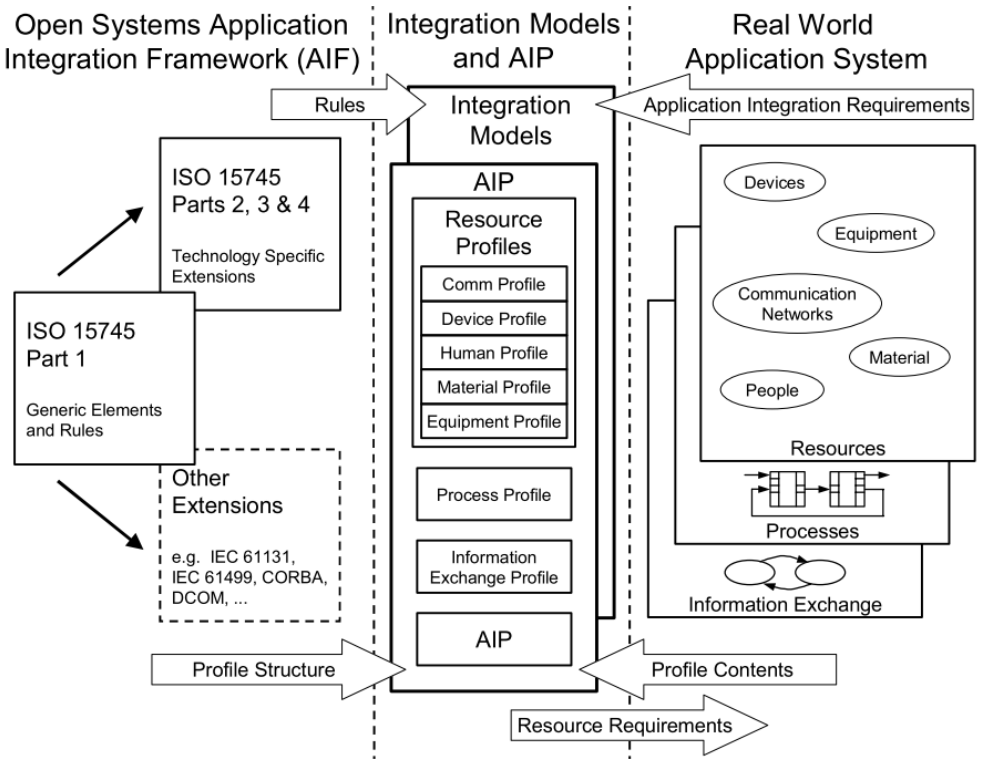


Figure 3.12: Context of ISO 15745 [55, Fig.1]

and references to other specifications. Many system integrators and end users who frequently operate in specific market sectors either generate multiple similar application specifications (one for each project), or generate a master application specification with variants for each project."[55]

The Application Integration Framework (AIF) defines elements and rules that facilitate:[55]

- The systematic organisation and representation of the application integration requirements using integration models
- The development of interface specifications in the form of Application Interoperability Profiles (AIP) that enable both the selection of suitable resources and the documentation of the "as built" application.

The contextual overview of this standard is shown in Figure 3.12. It represents the relationships of AIF (left), integration models and AIPs (middle), and real world application (right). The integration models are intended for capturing the application requirements, while the AIPs represent the available resource objects. During the system creation process these two are matched together.

This standard focuses on presenting information about an automation project in a platform-neutral format. The automation devices (like controller, bus terminal I/Os, sensor or actuator) are connected together with a network (i.e. fieldbus). The objective of the standard is to be able to represent any device connected into the control network. The standard functions as a container for user-specific information, which is appended by user-specified schemas. In this way they define profile, which actually is an interface specification. By comparing profile groups developed from different origins, one can match the application requirements (coming from

the application integration models as UMLs) with profiles of existing resources (XML). The application integration has three main properties: process, information exchange and resources.

The standard defines a schema only for high level abstract information (i.e. header template) and a container for more specific information, but XML schema (for, e.g. AIP) is completely left out and is for the user to define. Even though there are placeholders for different aspects of the production system, such as process, information exchange and resources, (which is further divided into communication, device, human, material and equipment profiles) none of the contents are defined. The communication-(network) related profiles are the only ones further specified in detail in parts two to five of this standard. Other specifications may use this standard as a container or vessel to convey their detailed information. One such link exists from the IEC 61804 Electronic Device Description (EDD) / Electronic Device Description Language (EDDL) (See Ch. 3.4.2.3).

3.4.2.3 Automation and Control Device Descriptions

There are quite a number of different module or component descriptions in the field of automation and controls. These descriptions usually represent a low level automation object like an IO-terminal, a valveblock, a servodrive, or a fieldbus-connectable sensor (e.g. position sensor). It gives information about the type and model of the device, memory mappings, size of data in and out, type of data, configurable parameters, and identification and vendor information. The description characteristics focus on fieldbus and control-related matters and needs. The descriptions are normally intended to serve the needs and specifications of a single fieldbus or a product family of a fieldbus, but there are few cross-domain descriptions.

The control device descriptions are mainly used in a development and configuration environments, which can read in the interface definition of the module for configuration, connection and communication purposes. These aid the application developer engineer when making the system configuration. Common characteristics of such electronic device descriptions are a text-based format basing on ASCII text, and internal proprietary structuring of the information. Examples of such automation device descriptions are:

1. Electronic Data Sheet (EDS) [56, A.4] used in ControlNet / DeviceNet networks.
2. Generic Station Description (GSD) [56, B.4.–B.6.] and EDD defines Profibus-connected automation components.
3. XML Device Description (XDD) is the file for Powerlink device descriptions. It relates to CANOpen.

These descriptions focus mainly on lower levels of detail (and entities) than the objectives of this thesis work. They could be applicable inside internal implementations of the production modules, but they can not fulfil the needs of external interfaces from these production modules. Thus, these descriptions focus on a different domain than this work, and are therefore beyond the scope of this thesis. However, it is worth giving them a brief analysis to identify what can be learnt from them and to see if they set any requirements and needs that should also be defined at the next encapsulation level.

A few automation object descriptions do have a wider and more generalised scope, and these are presented and discussed below. The discussion includes the analysis of their applicability to the concept proposed in this thesis.

Field Device Configuration Markup Language (FDCML) FDCML describes an automation component from various perspectives in a system-neutral format. Examples of described properties are identification, communication capability, functionality, diagnostic information, and mechanical description. Therefore, different applications can evaluate different aspects of a component. It can describe a range of different automation devices, from simple ones such as interface converters to more complicated ones, such as gateways with several protocol stacks. [154]

In contrast to other earlier device description languages, FDCML pursues the approach of a metalanguage. This means that devices do not have their own object models, but only generic structures are used to describe these objects. Then the generic structure can be used independently from the selected automation system and platform. The intended use includes: definition of profiles as a source of device development, source for device data sheets, and electrical device description as input for engineering tools. FDCML is used as the basis for the development of ISO 15745-3 [56] (See Ch. 3.4.2.2). It also relates to IEC 61499, IEC 61158, and EDD. FDCML version 2.0 is dated to 8.11.2002. [154]

FDCML is a system-independent device description language providing a generic overview and reference level description of the system of interest. It is a communication network and PLC related description of low-level components in an automation network. It has capabilities to express the system-level information of an automation project or installation.

Electronic Device Description (EDD) and Electronic Device Description Language (EDDL) EDDL and Field Device Technology (FDT) are presented and discussed in [62, 85, 104, 127]. The objective of EDD (IEC 61804-2) [49] and EDDL (IEC 61804-3) [54] is to define process automation devices, and to enable their easy integration into a control system. It provides interconnection and interoperability between any device, protocol, tool, or host with a single description. A device can be anything from a very simple sensor or valve to something more complex, like a servodrive. The description supplies information on three main categories: 1) 'Device Definition' including all the device parameters and description blocks; 2) 'Business Logic' including wizards (methods), conditionals, and maintenance and diagnostics functions, and 3) 'UI Description'. The Graphical User Interface (GUI) can include objects like menus, buttons and text boxes, trend and bar graphs, user-defined graphical objects and other visualisations. At the same time the EDDL file serves as data sheet for the device. The focus of the description is on the area of process automation, and it is independent of any one Operating System (OS) and automation system. [54, 155]

A contextual overview of EDD and EDDL standards is shown in Figure 3.13. EDD functions more as the methodology, architecture, and process, while EDDL is a structured and interpretative language for describing device properties. EDDL is used to describe the resources and to create the EDD file, which is then used with appropriate tools to generate interpretative code to support configuration, parameter handling, operation, and the monitoring of automation system components. The EDD processing has three stages: source generation, pre-processing, and compilation. The automation device manufacturer defines an EDD in the format of an EDDL and delivers it to the system integrator or user together with the physical hardware. The user interprets (pre-process) the EDD file in the format of their host system (Personal Computer (PC) or hand-held mobile device) or a certain application development environment understands. Thus, the single source file per device type can be utilised by different kinds of host systems operated in various OSs, which is a great advantage for both device suppliers and operators. The operator is able to use the same host environment for all devices, and there is a unified look and feel to the tool. The same applies to modules from different sources. Alternatively, the end user can develop a larger automation system application with the help of a development environment containing all the necessary links and connections, GUIs, etc. and finally compile the application from pre-processed information taken from the EDDs. The validity and neutrality of EDD is maintained by interpretation. There is no executable or compiled code within the description. This way the description remains up to date, and even host systems are updated or changed to a different one. However, the text-based approach neglects the issues related to installation and removal of drivers or SW. [54, 155]

The standard [54] specifies EDDL as a generic language for describing the properties of

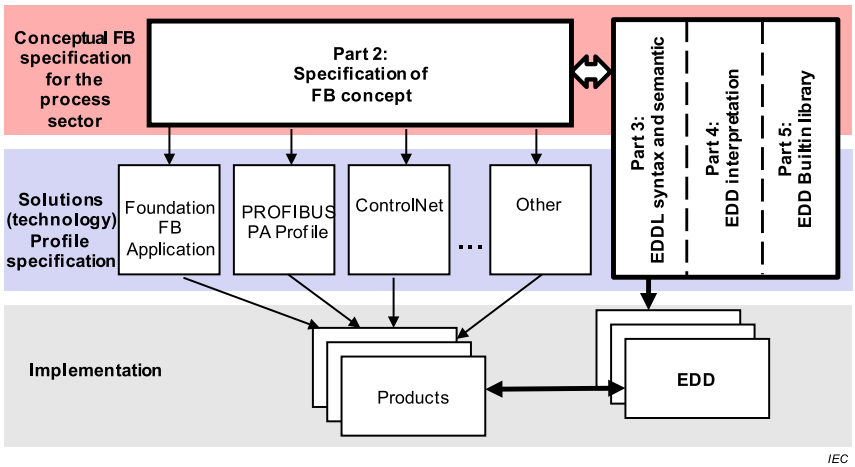


Figure 3.13: Context of IEC 61804 [54, Fig.1]

automation system components. It allows the manufacturers to use the same description method for devices based on different technologies and platforms. EDDL is capable of describing: [54]

- device parameters and their dependencies;
- device functions, for example, simulation mode, calibration;
- interactions with control devices;
- graphical representations:
  - enhanced user interface, for example, menus
  - graphing system e.g. trends
- persistent data store.

The semantics and lexical structure of the EDDL is defined in the standard [54]. It is mainly a descriptive language using ASCII, but it also allows conditions and methods similar to C -language. Therefore its accessibility is a tricky and specific tools are needed to process and interpret it.

The advantages of EDD and EDDL are the multi-link integration between devices and tools, with a single file-format. Included UI is a definite positive, as is the provided linking between the UI components' data and actions at the physical device, regardless of the communication channel or protocol used between the parties.

The main drawback of EDD is that it supports only a few fieldbus systems mainly utilised in process automation (HART, Foundation Fieldbus, and Profibus). This limits the opportunities for the discrete manufacturing domain. Even though the target is that EDDs would be interchangeable, each fieldbus system still requires its own kind of EDD file. Both of these issues are clear limitations in the context of this thesis. Another issue is the complexity of the used EDDL. It is pure text appended with C -code. It misses the clarification offered by hierarchical structuring as all the information is presented more or less as a flat list. This makes maintenance a bit more difficult and increases the need for a specific tool for the maintenance of the stored information.

Open-EDDML Pantoni and Brandão [103] have been proposed as an open alternative to EDD and a more accessible format for EDDL basing on XML, rather than the C -language type of approach [103, 104]. The language they have developed is called Open-EDDML (Open Electronic Device Description Markup Language). It more or less represents the semantics of EDDL in an XML syntax. Therefore the content of the files are easier to access and process by a larger number

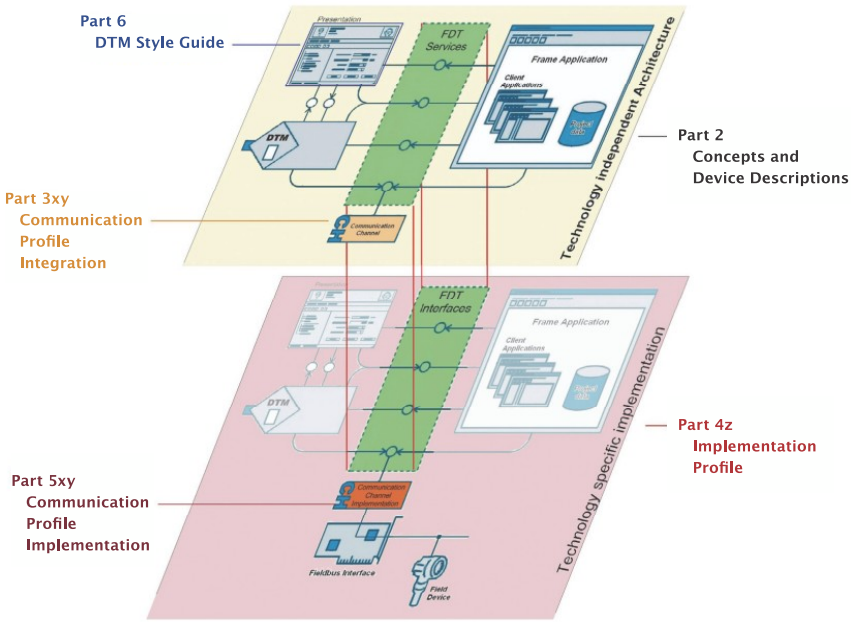


Figure 3.14: Basic Structure of FDT [156]

of tools or SW components. They have also developed a set of open tools to process and use the files according to the model defined by EDD (See Ch. 3.4.2.3).

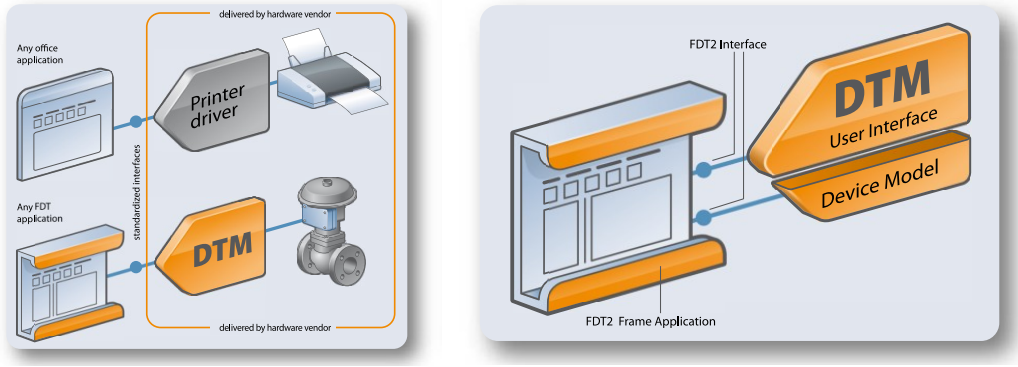
Field Device Technology (FDT) The acronym FDT is associated with various terms in the literature like Field Device Type, Field Device Tool, or Field Device Technology, which is the term used here, as it seems to be the most recent one. [32, 33, 156]

FDT standardises the communication and configuration interface between field devices and host systems (e.g. control system, or engineering or asset management tools). It provides a common environment for accessing the devices' most sophisticated features. Any device can be configured, operated, and maintained through the standardised user interface - regardless of supplier, type, or communication protocol over different network hierarchies. [32, 156]

The FDT specification already has a long history. It is a further development of EDD and EDDL (See Ch. 3.4.2.3). The development was initiated in 1998, followed by the first major versions v1.2 in 2001 and v1.2.1 in 2005. The specification was later standardised as IEC 62453 and ISA 103. The current version 2.0 of FDT was released in 2012. [32]

The specification has two key elements: a) Device Type Manager (DTM), and b) FDT Frame Application - the 'Host system'. These are connected together by the FDT interface and architecture (See Figure 3.14).

The Device Type Manager (DTM) is a device driver supplied by the device manufacturer. It operates like a printer driver does in a PC environment (See Figure 3.15a). DTM provides a unified structure for accessing device parameters, configuring and operating the devices, and diagnosing its problems. It contains data, functions, logic, and GUI elements of the device. DTMs can range from a simple GUI for setting device parameters to a highly sophisticated application capable of performing complex real-time calculations for diagnosis and maintenance purposes. The device manufacturer supplies the DTM together with the device, including device-specific



(a) Analogy between printer driver and FDT

(b) FDT Frame Application and DTM

Figure 3.15: Examples of principles and structures of FDT [32]

functions, and provides a user interface as a driver for integration in a Frame Application. [32, 156]

The main advantages are in the reduction in effort needed to integrate the devices into heterogeneous systems. The operators can operate devices from a single FDT Frame Application that offers a similar look and feel to all devices in the factory, however they are connected. The device manufacturer needs to provide only a single device driver that can be utilised by any Frame Application. The access to devices is unified, so that manufacturers do not need to change a thing when there are changes in the system environment, like topology or protocol changes.

The DTM is composed of two separate sections, which are supplied by the device manufacturer. These are (See Figure 3.15b): a) Device Model or Business logic, and b) User Interface. The Device Model is responsible for processing data and encapsulating device-specific and protocol-specific functions. With the help of the device model, the FDT Frame Application is capable of interacting with any device or protocol without knowing their specific details. The Device Model is used for running the functions like communication with devices, browsing the FDT topology, saving data, or interacting with DTM user interface objects. [32] DTM User Interface establishes user access to the device through DTM GUI and its functions. These provide the user with a possibility to see or change device parameters or to execute other functions in the device.

The GUI is a tool and access point to operate the devices on a daily basis. Thus, it has a substantial impact on the efficiency of a system and is well appreciated by the operators if it is easy to use and provides uniform structure across all manufacturers and device types. Therefore, FDT defines a DTM Style Guide that outlines the rules for the structure (look and feel) of the user interface. It prescribes the division of the user interface into general and task-related areas, provides a library of icons and their meaning, and provides a glossary of terms and standard messages in several languages. Thus, users encounter the listed components in all DTMs with the same appearance and the same meaning. [32]

There are two types of DTMs: 1) Device DTM which represents a physical automation field device, and 2) Communication DTM which connects the software communication components to physical network. The latter is further divided into two 2.a) Communication DTM Interface used as the primary access point to the FDT network, and 2.b) Gateway DTMs which are devices connecting together two different networks with different protocols. In addition, there are Interpreter DTMs, which are interpreted from other descriptions like EDD or EDDL. Usually these have a limited functionality. [32]

The Frame Application is a software program that contains and implements the Device DTMs and Communication DTMs and it provides them with a runtime environment (See Figure 3.15b). At the same time, the Frame Application provides interfaces for interactions between the device model and the user interface and their physical environments, as they are not allowed to interact directly with each other. It hosts GUIs defined by the DTMs and provides applications for a common execution environment; management of users, DTMs, and data; network configuration, and navigation. The Frame Applications provide fieldbus-neutral access by supporting various communication protocols. It guarantee a system-wide, uniform configuration method and allows multi- or single-workstation environments. The frame application can be integrated and embedded as a unified communication channel to the factory floor into different applications like PLC, OPC Unified Architecture (OPC UA) server, configuration or engineering tools, console terminals, or asset management tools. [32, 156]

It seems that the main objective of FDT is in the configuration, commissioning, monitoring, diagnostics, and maintenance of automation field devices in the domain of process automation. There is, however, not much support for system design. It is intended to connect and configure devices in heterogeneous environments by providing an easy and generic method to access and configure the automation devices. The main reasons for its success and excellence are: the device driver and GUI windows are provided by the device vendor; support of various communication methods (fieldbuses) and hiding their variety and complexity from the application; easy integration of devices into an application framework without any additional coding, and built-in support for proprietary access and communication methods to the devices.

The openness of the FDT is jeopardised if a *Microsoft .NET* environment is used as the application platform and object-model, even though it is widely supported. The risks are similar to what they were with OPC before they made the change into more open technology with OPC UA. Although the FDT contains parameters, the description is not intended for direct support for the design and selection of system components. Instead, it focuses more on supporting and solving issues during daily use and the parametrisation of the device, and providing a lower-level communication channel to the device. Therefore, despite its many charms, it does not fulfil the needs presented in this thesis. However, it might well be a complementary specification for GUI-sharing in the future.

Field Device Integration (FDI) Field Device Integration (FDI) technology is a recent development which combines the advantages of FDT with those of EDDL in a single solution. FDI takes account of the various tasks over the entire life-cycle for both simple and the most complex devices, including configuration, commissioning, diagnosis and calibration. The drawback of FDI technology, in addition to the ones listed in the case of FDT, is that it is even more focused on the process industry domain than even FDT. It supports only three fieldbus systems, Fieldbus Foundation, HART, and Profibus / ProfiNet.

3.4.3 Process and Logic Descriptions

The *Process Specification Language (PSL)* [57, 157] creates a common ontology between N supply chain partners. The aim is to provide a middle-level exchange layer, so that every connecting partner needs to implement only one single, common interface to communicate with others. Therefore, PSL creates mapping between the different terminologies used by either side through this shared concept at the centre. Partner *A* creates their own terminology and ontology which is then mapped to the common PSL ontology. In doing so, the specific process, e.g. quotation, a part of manufacturing process, or the logistics, can be expressed by Partner *A* with PSL. Later, it can be sent in the other direction to the receiver side (Partner *B*), who then responds, vice versa. Although PSL cannot be used for describing resources or production modules per

se, it can be used to generally define different processes in a common ontology. However, its applicability for the objectives of this thesis is limited.

XML Process Definition Language (XPDL) is a format standardized by the Workflow Management Coalition (WfMC) to interchange business process definitions between different workflow products, i.e. between different modelling tools and management suites. XPDL defines an XML schema for specifying the declarative part of workflow / business process.

3.4.4 Controls-related Descriptions

Other sections have already presented a few descriptions and technologies from this category. For example, Ch. 3.4.2.3 gives a broad set of automation device descriptions for control application development.

PLCopen is an organisation for standardising PLC domain-related matters. This includes programming languages like the IEC 61131 [52]; function blocks for motion control, so that application programming is harmonised across the PLC providers; communication with close collaboration with OPC Foundation (OPC UA [158]), and PLC program exchange.

PLCopen XML [159] is intended to complete the PLC program exchange, and it is specified by the PLCopen TC6. It is an open, vendor-neutral transfer format between different PLC development environments, providing both export and import features. Even though this intermediate transfer format is defined as an interface for IEC 61131-3 [52] -based SW tools, it can be utilised for various other purposes. These include other environments and tools, like visualisation and HMI, project design and documentation, configuration tools, debugging, simulators, networking tools, and version control, but is not limited to these. One special use case is to distribute logic libraries. It will greatly reduce the necessity to re-type or re-engineer information, as it is not limited only to textual information and engineering design (the logic), but also includes graphical information. It is integrated as is the sequencing definition format for AutomationML (See Ch. 3.4.1.1).

OPC Unified Architecture (OPC UA) is a platform-independent standard for communication of industrial automation devices and systems documented by IEC 62541 standard. OPC UA is an industrial machine-to-machine, machine-to-business, and business-to-business data exchange and communication protocol for interoperability. It was developed by the OPC Foundation and released in 2008. Their intention is to provide a cross-platform Service Oriented Architecture (SOA) for process control, while enhancing security, and providing a common information model that would be at the same time future-proof, scalable and extensible. This differs significantly from its successor OPC, which is based on Microsoft's communication model Component Object Model (COM) and its distributed version DCOM. The antecedent OPC is currently called the OPC Classic. [158]

The OPC UA is intended to provide platform independence both in HW and OS, which increases its applicability for interoperability. It adds features for discovery, method calls, and different models for event-based communication like subscriptions and notifications. Security aspects are included, such as encryption, authentication, user control, and auditing. It offers increased reliability of information transfer over various media and protocols, including retransmission in the case of a lost connection. Most important of all, however, is that its common framework and information models, which are divided into selectable information modules, provide the foundation for this mechanism of communication. [158]

A recent development (starting from February 2015), called *Vorto* [160], provides information models for Internet of Things (IoT) devices, including definitions for the capabilities, functions, data types and interfaces. In addition, it provides a concept and associated architecture (illustrated in 3.16) to model, distribute and utilise these device descriptions. The architecture also includes a harmonised tool chain. The information model is divided into abstracted and device model parts.

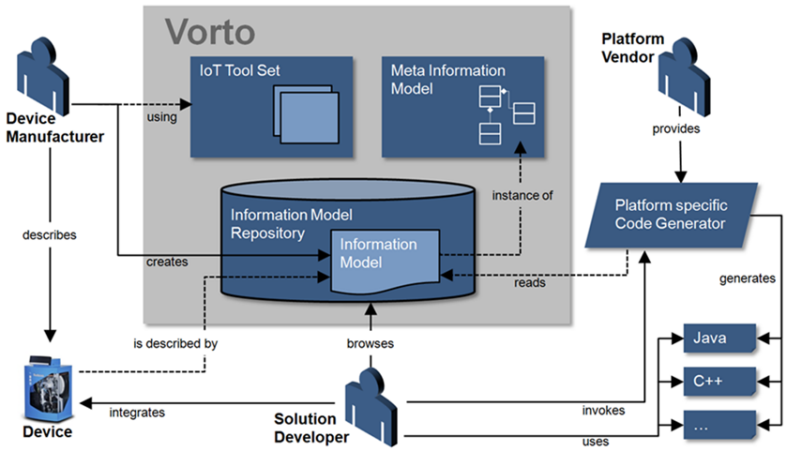


Figure 3.16: Overview of the Vorto architecture [160]

The 'Device Manufacturer' provides the 'Information Model' of their module, and publishes it through the 'Information Model Repository'. The 'Solutions Developer' searches for suitable devices for their system from the repository, and finally invokes the 'Platform Specific Code Generator', that generates solution-specific code from the published information models. The code generator is also a common and shared resource with the 'Platform Vendor'. [160] Thus, there are a lot of similarities with the concept for interoperability proposed in this thesis. However, the main difference is that Vorto is strongly focused on the controls domain and how to integrate controls from different sources together. The scope of this thesis is wider, and encompasses the module's characteristics and the extent of the interfaces. Any deeper evaluation and possible integration of *Vorto*'s control concept is left for future work.

A few *SOA*-related modelling languages are included in this analysis. The *Service Modeling Language (SML)* [145] provides a rich set of constructs for creating models of complex services and systems, and models them as a collection of XML documents. It mainly focuses on the IT domain, modelling the IT resources, services and their interrelations. Depending on the application domain, these models may include information about configuration, deployment, monitoring, policy, health, capacity planning, target operating range and service level agreements. The SML model aims for modularity, re-use, standarsisation and validation, and it has two parts – the schema and rule parts. The associated *Service Modeling Language Interchange Format (SML-IF)* [144] provides an implementation-neutral interchange format that preserves the content and interrelationships of the SML documents. [144, 145] The SML has a strong focus on the IT domain, and it provides dynamically an integrated model made out of many model fragments, which can come from various sources. Thus, the single schema per model approach sounds simpler.

The *Service oriented architecture Modeling Language (SoaML)* [96] is an open source specification from the OMG, and it includes a metamodel and a UML profile to specify, model and design services within a service-oriented architecture. The SoaML supports the following modelling capabilities: a) identifying services, their requirements and interdependencies; b) specifying services including functional capabilities, consumer expectations, the protocols and information exchange patterns; c) defining service consumers and providers; d) the policies for using and providing services; e) classification schemes for the services, and f) integration with OMG's other models, such as the Business Motivation Model and UML. [96] The SoaML can be applied in

cases where module controls are implemented as SOA services. Its applicability to model the other aspects of the production resources remains weak.

3.4.5 3D Model - Geometry and Kinematics

The resource description needs to represent the geometric model and kinematics of the production resource. There are several different models and representations, such as STEP, COLLADA, eXtensive 3D (X3D) and various Computer Aided Design (CAD) formats. The resource description should re-use these formats and reference them, and not to try to make yet another definition. Thus, these are not presented here in detail. Only two descriptions, mainly used to model the kinematic structures, are brought up here. These are the Unified Robot Description Format (URDF) and the Denavit-Hartenberg parameters.

3.4.5.1 The Unified Robot Description Format

The Robot Operating System (ROS) [161] and ROS-industrial [162] offer a framework for robotic applications. They provide a Hardware Abstraction Layer (HAL) for robotics, so that a robotic application can be created from a distributed and modular design, and from reusable components which do not mind what kind of robot they are interacting with. The ROS is based on message-passing middleware (publish and subscribe) and numerous plug-in type of modules. These provide additional functions such as modelling and visualising the robot, moving them, recognising and interacting with objects, collision detection and path planning, navigation, and diagnostics. The ROS also includes a description for modelling the robot and its kinematics in a machine-readable way. [161]

The ROS's robot description language is called Unified Robot Description Format (URDF) [114]. It is an XML document, which describes the physical properties of the robot, such as the lengths of the links, the sizes of its wheels and body, the locations of its sensors, and the visual appearance of each part of the robot. It contains the kinematics, inertia, visualisation, and simplified collision modelling-related information for the links. In the case of the joints, associated links are defined; type of joint; limitations for movement in position, velocity, and force; dynamics properties, and calibration. Other applications within the ROS framework can then utilise this information about the robot, especially in the case of transformations and visualisations. Important to note is that there is a conversion from URDF to COLLADA.The first version (0.1.0) dates back to 2009 and the current one, (1.11.5), came out on July 2014. [114, 161]

The general impression is that URDF is well suited for simple representations of a robot model, including the body (in rough 3D form), its kinematics, and a set of its dynamic features. However, that is all it is good for. It does not tackle the interfaces, i.e. mechanical interfacing with the robot, nor any of the other properties required for selection of a robot for a production system. However, it could well be used as part of a module description, representing the kinematic body of the device. It could be used either as the main or a parallel description for kinematics inside the proposed Module Description. It is an already-existing specification, and there is a set of tools for it. ROS and URDF are getting more and more accepted by the research community, and since the arrival of ROS-industrial, also (slowly) by industry.

3.4.5.2 Denavit-Hartenberg parameters

The Denavit-Hartenberg (DH) parameters represent the kinematics of bodies in 3D space. It is not a description language, but a well-known and accepted method for expressing kinematic structures. In this thesis, the definitions made in [66, p.233..380] are followed. The DH parameter set contains four parameters associated to a single link (L_i) of a kinematic structure. These

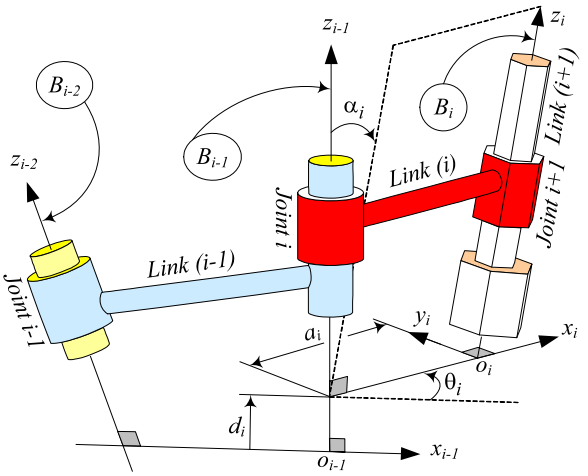


Figure 3.17: DH parameters $a_i, \alpha_i, d_i, \theta_i$ defined for Joint i and Link (i). [66, p.236]

are: link length (a_i), link twist (α_i), joint distance or link offset (d_i), and joint angle (θ_i). [66, p.233..236] explains how to assign the frames and determine the values for these parameters. The notation of links and the associated parameters are illustrated in Figure 3.17. The proposed Module Description utilises this method and stores the information according to this methodology.

3.4.6 User Interface Descriptions

This section collects together device descriptions that contain a UI component. The UI of the device is prepared by the manufacturer and delivered together with the device. It is later used to visualise the device by runtime, or another kind of environment, after the device has been integrated as part of the larger system. Such an arrangement will save a lot of time as the system GUI can be prepared from ready-made components. The device GUIs can be utilised and taken advantage of during the commission, configuration, ramp-up, monitoring and maintenance phases of a production system.

The main interest here is on specifications for an abstract meta-language that can provide a canonical XML representation of any GUI. Souchon and Vanderdonckt [128] have compared and analysed a few such UI Description Languages, along with their properties, applicability and maturity. González Calleros et al. [42] discusses different kinds of frameworks for model-based UIs. They present a reference framework for UI abstraction levels and then compare different implementations related to these identified abstraction levels.

The language should be capable of specifying the requirements, design, and implementation of any UI. This would mean that the manufacturer of a device can use the language to describe and define the functionality of the HMI in a common format. In the phase of execution, this common language definition is compiled or interpreted by the target environment, whatever it is. By use of a neutral and common format, the options for different implementations will remain open, allowing different platforms (in terms of OS, architecture and SW applications) to be used, and paving the way for future developments.

However, special care is needed for connection and interaction with the physical HW, and how the physical IO signals are linked and connected into the HMI. This needs careful evaluation, especially because of the heterogeneous target environments and the need for compatibility with

the control methods. A few already-discussed description formats fall into the category of UI descriptions, or they contain properties and features of value to the UI aspects. These are:

1. EDDL includes GUI descriptions. (See Ch. 3.4.2.3 (p.44))
2. FDT includes GUI descriptions and implementation of remote screen applications. (See Ch. 3.4.2.3 (p.46))
3. FDI contains sections that focus on GUI descriptions and applications. (See Ch. 3.4.2.3 (p.48))

In addition to this, there are particular GUI descriptions, such as User Interface Markup Language (UIML) [87, 136, 163], USer Interface eXtensible Markup Language (UsiXML) [140, 164], eXtensible Interface Markup Language (XIML) [107], and Model-based lAnguage foR Interactive Applications (MARIA XML) [107]. Similar concepts exist in other fields, such as agriculture, ISO 11783-6 (ISOBUS) [95, 165].

It is noticeable that many of the particular GUI description formats have not yet been standardised. Many of them have been inactive for a long time, and have few associated publications per description. The tools supporting them have not improved, either. One could even say they are vanishing. However, even if the approach is valid, attractive and beneficial for production devices operating in the context of a heterogeneous environment, it is excluded this stage and left for future work. This is mainly because of maturity and the lack of sufficient tools.

3.4.7 Summary of the Description Languages

Figure 3.18 maps the relationship between the various descriptions collected in this literature review. The category of each description is illustrated with colour. The blue ones are module-level descriptions, which express information at the same level of interest as this thesis. That makes these descriptions the most interesting and important fruit of this literature review. The green ovals represent descriptions expressing system-level information. The orange ones are descriptions representing 3D and/or kinematics information, which can be used for displaying a graphical model and/or kinematic behaviour of the device. Violet is used to represent the process languages, and finally, the yellow ones are descriptions focused mainly on either controls and/or communication, although in some cases the description falls into several of the previous categories. These secondary categories are indicated with the star(s) associated to the oval, which follow the same colour scheme as the main categories. There are a few different kinds of associations, which details the types of relationships. The main relational link is either uses/utilises or 'derives to'. This is indicated with the unbroken line and arrowhead. The weaker link is represented with a broken line. The inheritance between the two descriptions is expressed with a UML inheritance symbol, as is the aggregation.

Appendix A / Table A.1 summarises the main aspects of a few of the most relevant description languages in the field of this work. Each row represents a single description, in which is presented the name and/or acronym as identifier, short description and purpose, is the description based on other definition(s), what is the standardisation / developer organisation behind the description and reference to the chapter presenting the specification. Finally, and perhaps most importantly, an evaluation of the issues of the specific description is discussed.

It is noticeable that many of the descriptions which were identified as being well-developed and as having properties and features suitable for the objectives of this thesis are from the process automation domain. There are two reasons behind that. The first is the heterogeneity of the domain, and indeed, the business. There are many companies supplying specific technology, like measurement devices and sensors for flow, pressure, level, etc. One provides the valves while another makes the controllers. In the end, all of these devices need to be integrated together into one operating system. This highlights the importance of easy integration and interoperability. Because there are so many players involved, and a few industry giants do not control the entire scene from the bottom to the top, as is beginning to happen in discrete manufacturing, the

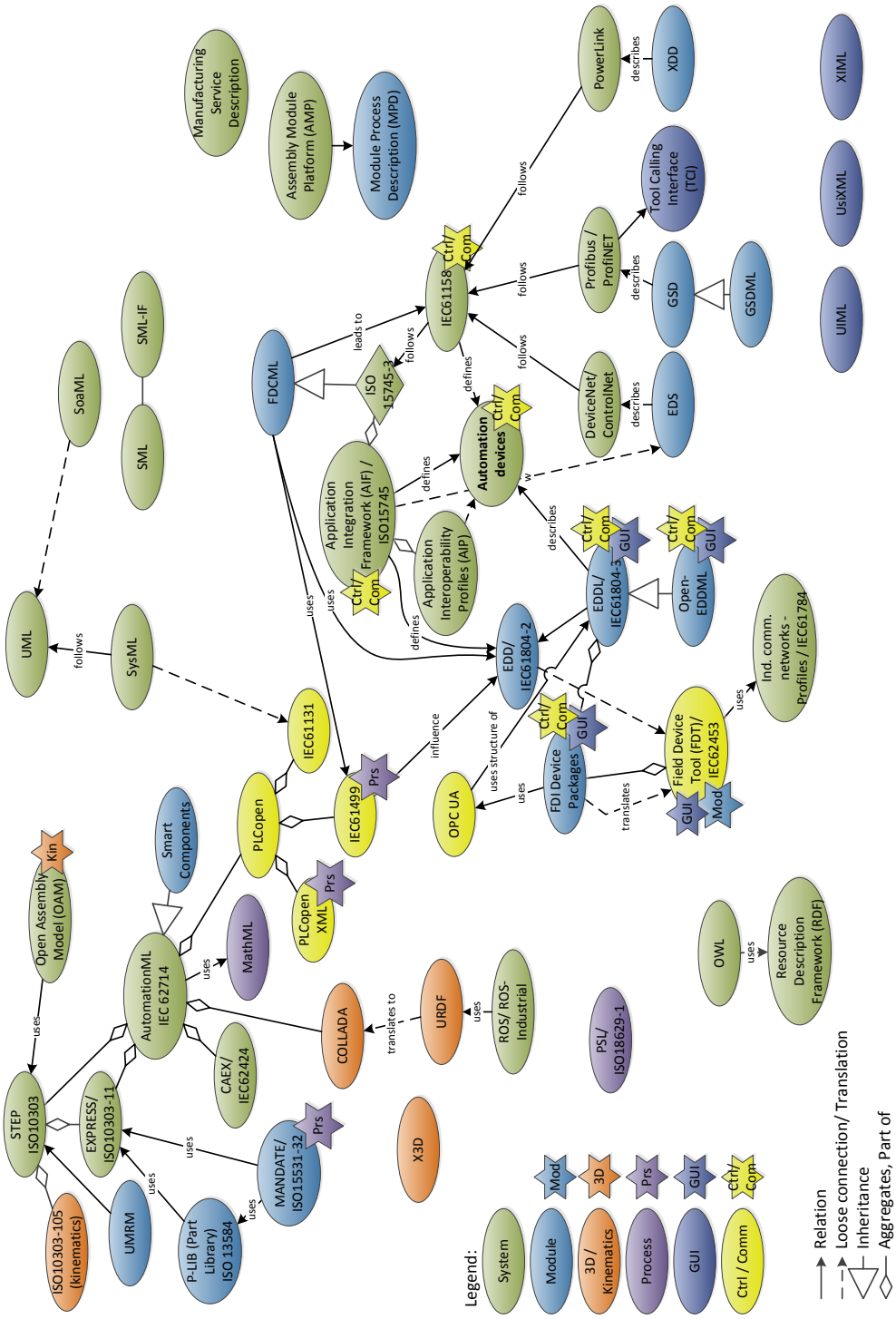


Figure 3.18: Relations map of most relevant description languages

companies involved have been forced to unite their interfaces and interfacing technology for the common good. The second reason is that the response time can be slower in process automation than in factory automation – roughly one second or more for the process automation, versus ten milliseconds or less for the latter. The slower response requirement allows different technological solutions to be used. As in the case of FDT or FDI, the single message exchange is routed and jumped through several layers, both in the virtual and physical worlds, which naturally slows down the response times. But as said, this is acceptable and convenient for the domain, providing ease in integration and operation, and response to system or topology changes, which are obvious and unbeatable benefits.

Another conclusion drawn is the levels used in the approaches and descriptions. There are both bottom up and top down approaches, which then drill in towards each other as best as they can until the saturation point of applicability is met. There is no sense to continuing to apply the description after that line, because, after crossing it, the description's power and practicability to define the phenomenon is weakened or lost.

Many of the device descriptions start from the low levels of integration in controls. A good example of this are the fieldbus communications, and how their devices are connected and configured into the communication network (See GSD, EDS, or XDD). These developments were necessary in order to prepare the foundation for later integration. Subsequently, the descriptions have moved on the next level, which is the case with the descriptions in EDDL, FDT and FDI. They define a process device, how it can be connected into and accessed from the communication system(s), and how it can be integrated and operated from the control system. The HMI is included as well.

Another approach is from top down. This is illustrated by descriptions like AutomationML, STEP, SysML, etc. These are effective for describing and capturing a product or a system, but they are to follow the project engineering approach, for more or less one-of-a-kind systems. They start from scratch and go on until a new system is created. These end up being too high level to be implemented at shop floor level, even though a few, such as the AutomationML, are getting pretty close. Still the problem of designing and commissioning an RMS-type system (systems built from re-usable modules) in an effective way remains unsolved.

There is a gap between the two approaches (bottom up versus top down) which indicates a need for yet another layer of abstraction. This should focus on defining the integrated process modules possessing capabilities. It will require a more complete and comprehensive view from the system design perspective conducted from product requirements, and should unite the control aspects at a higher level.

The coverage and positioning of various specifications from the literature discussed here are illustrated in Figure 3.19. It positions the specifications on a two-dimensional matrix. The horizontal axis represents the major stages of a product's or system's life-cycle. The vertical axis represents the three different viewpoints of the information. The aspect of resource is added to the original set for the reasons presented in Ch. 3.2.2, Figure 1.1 and [12], and the viewpoint of 'Enterprise services' is omitted as it is not applicable. Specifications with similar coverage are bundled together under the same oval for clarity and space-saving reasons.

The objective of Figure 3.19 and Table A.1 is to show and identify the existing gap. None of the current definitions is able to thoroughly describe the production devices for a discrete manufacturing domain so that the information provided would support the various production system design phases. This is especially true for the RMS-type systems, which are designed from existing and re-usable modules.

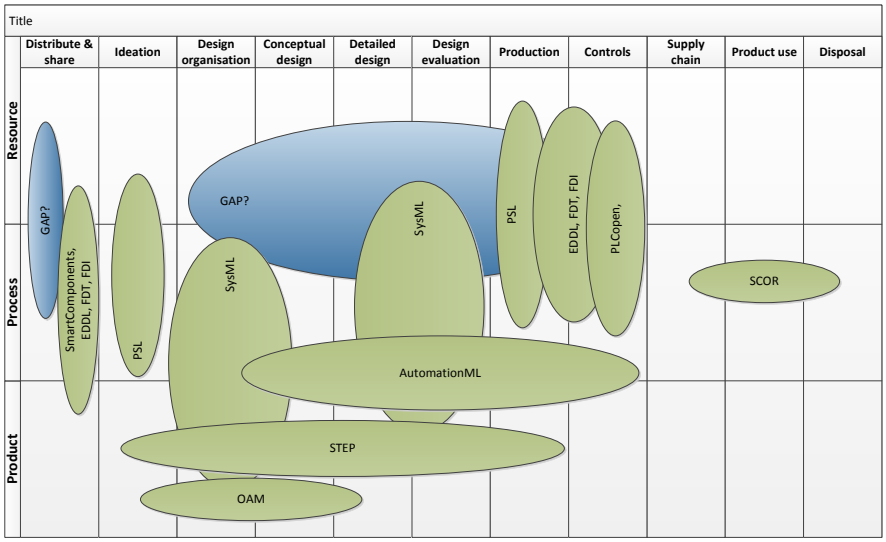


Figure 3.19: Specification and their promised coverage. Modified and extended from [108]

3.5 Associated Work from Others Related to the presented Framework

There are a few other contributions that can be associated with the sections of the framework of the production system design process presented here (See Figure 5.10 (p.98) and Figure 5.11 (p.102)). These works have been collected into Table 3.2 and Table 3.3. The first three columns presents the author, title of the thesis, and year and month of publication. The last column maps the item to a specific process (alphabets) and/or document (numbers) in the presented framework.

Table 3.2: Other PhD Theses associated to the framework of production system design

Author	**Title**	**Date**	**Association (Figure 5.10)**
Barata, Josè (UNINOVA)	Coalition Based Approach for Shop Floor Agility - A Multiagent Approach [6]	2003	E, F
Lastra, José (TUT)	Reference Mechatronic Architecture for Actor-based Assembly Systems [83]	2004-05	10, 11, E, F, 13
Lohse, Niels (UNOTT)	Towards an Ontology Framework for the Integrated Design of Modular Assembly Systems [75]	2006-05	A, B, 11
Gershenson, Carlos (VUB)	Design and Control of Self-organizing Systems [41]	2007-05	F
Lanz, Minna (TUT)	Logical and semantic foundations of knowledge representation for assembly and manufacturing processes [73]	2010-07	A, B, 11, 13
Frei, Regina (UNINOVA)	Self-Organisation in Evolvable Assembly Systems [37]	2010-08	E, F
Ferreira, Pedro (UNOTT)	An Agent-Based Methodology for Modular Assembly Systems [34]	2011-04	2, B, 3, F, (A, C)
Ribeiro, Luis (UNINOVA)	Diagnosis in Evolvable Production Systems [113]	2012-02	F, E
Järvenpää, Eeva (TUT)	Capability-based Adaptation of Production Systems in a Changing Environment [65]	2012-11	B, 11, 13, 3→B
Maffei, Antonio (KTH)	Characterisation of the Business Models for Innovative, Non-Mature Production Automation Technology [78]	2012-12	B, C
Neves, Pedro (KTH)	Reconfiguration Methodology to improve the agility and sustainability of Plug and Produce Systems [93]	2016-05	B, E

Table 3.3: Other work associated to the framework

Author	**Title**	**Date**	**Association to framework (Figure 5.10)**
-	AutomationML [3]	2013	3, 4, 12, 2(partial)
-	IEC 61512 - Batch Control (or ANSI/ISA S88) [48]	1997	2, 3, 4, 10, 11
-	IEC 62264 - Enterprise-Control System Integration (or ANSI/ISA S95) [51]	2001	3, 4, 10, 11

4 Definition of the Requirements for Module Descriptions

This chapter is based on the findings of the literature review; especially the production system design principles presented in Ch. 3.2 and the targeted production systems described in Ch. 3.3. After a brief introduction to the development environment and the two case studies used for development and verification, the chapter defines the User and System Requirements. The generic level requirements are defined first, followed by the identification and definition of the users in this particular context. Each user category is analysed with use cases, from which the User Requirements can then be defined. The chapter concludes with a description of the System Requirements.

4.1 Introduction to the Development Environment

The development environment has multifunctional roles in the context of this thesis. Firstly, it is used for both data and information gathering. Secondly, it serves as a platform for the physical and Hardware (HW) modules to play with, and thirdly it is used to model the devices. The sources used to formulate the User Requirements are documents and reports, face-to-face meetings and, in particular, workshops, as well as a number of interactive discussions with individual users. Finally, there are the physical devices themselves.

The core of this work has been developed and evaluated with two different case studies. These are related to the physical modules and how those modules are described with the developed description models. The first case study, *Case 1*, is actually two chronologically separate versions of a real-life production line dedicated for research purposes. The first version, EUPASS Platform - Version 1 was built at the University of Nottingham (UNOTT) in the autumn of 2006, while the second version, EUPASS Platform - Version 2, was built at the University of Applied Sciences of North-western Switzerland (FHNW) [166] two years later and contains over twenty HW modules. This case study is used to develop and evaluate the concept proposed in this thesis. It contains a number of development iterations, as defined in Figure 2.2. The Case 1 environment is used firstly to develop the proposed concept and its associated methodology, and then for verification. Case 1 mainly focuses on providing a platform for the iterative development of the proposed concept and descriptions, and serves as a testing ground for real application requirements, and for verification of the functionality of the developed structures.

The other environment, *Case 2*, is the TUT microFactory (TUT μFactory) built at Tampere University of Technology. It will serve as a validation and verification environment for the developed methods and descriptions. The verification phase is discussed in Ch. 8.2. The workshops, conference papers and presentations are also used as forums to validate the proposed concept.

4.2 User Requirements

This section defines the User Requirements (URs) which the developed and proposed Description concept, methodology, and implementation are designed to answer. These URs have been collected from a variety of sources, particularly the meetings, workshops, interviews, discussions and documentation used during and after the EUPASS project.

4.2.1 General Requirements

In general, the concept has to show the following capabilities: a) exchangeability and interchangeability, interoperability, compatibility and modularity enabling the re-use of components at both the HW and Software (SW) levels; b) openness; c) portability of the platform, architecture, Operating System (OS), programming environment and other SWs; d) independence and autonomy, and e) exactness, unambiguity and ease of use. The terms used for these key requirements are comprehensively defined below within this context.

Exchangeability is defined in [167, term: Exchange] as "an act of giving one thing and receiving another (especially of the same kind) in return." The related term, **interchangeability** is defined in [168, term: Interchangeability] as "The ability of a system or product to be compatible with or to be used in place of other systems or products without special effort by the user", which best suits the purposes of this thesis.

Interoperability is defined in [168, term: Interoperability] as "the ability of a system or a product to work with other systems or products without special effort on the part of the customer. Interoperability is made possible by the implementation of standards." [86] discusses interoperability in different contexts, and why and how to pursue it. As well as providing a definition, [169] lists two approaches for achieving interoperability, either or both of which can be present at the same time. The first approach is to adhere to published interface standards, and the second is to use a broker service or adapter that can convert one product's interface to another product's interface 'on the fly'. **Compatibility** is closely related to the term interoperability in that a product can be compatible with a standard, but interoperable with other products that meet the same standard (or can achieve interoperability through a broker) [169].

Both concepts, i.e. Interchangeability/Exchangeability and Interoperability, are the main reasons and drivers for this work. The objective of the concept presented here is that it enables interchangeability between the different components used in the production system. This ensures that the functionality of the system is maintained throughout, regardless of any changes taking place in the system. In other words, the components are interoperable with each other across different system components (type, variant, etc.) and across different vendors.

Openness and **Portability**. This concept is in accordance with Open Architecture [92] principles of, for example, Open Architecture System or Open Architecture Controls. The concept and file formats must have neutrality and interoperability across different platforms, architectures, OS, programming environments and other SW. The concept shall be applicable across platforms without any limitation or system locking. However, it should be noted here that the solutions do not necessarily follow the principles of "Open Software", or more precisely "Free and open source software", and Open Source Definition [170].

Modularity and **Re-use**. The production system, the description concept, (and all the applications utilising the concept) should all promote and utilise modularity and re-use as much as possible in order to support and facilitate maintenance and prevent errors. These are all partial enablers of sustainable manufacturing, and given a high-quality production module, they will enable a long life-cycle for a module. As long as the production module is of a sufficiently high quality in terms of performance and precision, and established interfaces are used, a module need

not become prematurely obsolescent. The philosophy '*Define only once, re-use multiple times and places*' should be followed from both the descriptions and SW points of view.

Independence, **Autonomy**, and **Self-contained**. Each description (file) shall be as autonomous and self-contained as possible, and its essential parts should be usable without any additional information sources, such as external sources from the internet or from a Database (DB) connection. The core information in the files must be fully inclusive, i.e. when someone receives a file it must contain all the necessary information about the features and properties of the module and its interfaces.

Exactness, **Unambiguity**, and **Clarity**. The information presented should be exact, explicit, and unambiguous. There should be no room for misinterpretation during the design and utilisation phases. These are important factors for error avoidance during utilisation, and in order to promote the credibility and prompt acceptance of the presented concept. **Internationalisation** is closely associated with these three terms. All the structural entities of the description language and the common parts of the module descriptions have to be written in English, which is the default language for the module descriptions. In addition, the capability of internationalisation should be offered in those sections of the module descriptions which are intended to be read by humans, such as labels, documentation, instructions, and Graphical User Interfaces (GUIs) in general.

Ease of use and **simplicity** are important features. The more the concept is able to display these qualities, the greater the level of widespread acceptance and adoption. Success in these areas will determine how well the concept is accepted at shop-floor level and in everyday business practice.

Data integrity and **Intellectual Property (IP) protection**. The concept and associated files should be secure and must provide mechanisms for ensuring data integrity. This means that any data corruption or tampering with the descriptions should be at least recognised and at best prevented altogether. Although the descriptions have to be open and transparent in the outer shell of the module, it must be possible to hide the core IP (e.g. the implementation and technological advances) of the module. In other words, black-box [171, term: black-box] or grey-box [171, term: gray-box] thinking must be behind the implementation, which is an important aspect, especially for Small and Medium Enterprises (SMEs) as it enables open and true competition between products (i.e. production modules). This is an important (possibly the most important) enabler for a truly multi-vendor integrated production system.

4.2.2 Context-related Requirements

There are a number of requirements associated with the context of a manufacturing domain, specifically to a Reconfigurable Manufacturing System (RMS) and an Evolvable Production System (EPS). These terms are listed and discussed below.

- **Machine readable and writeable**. The content of the files for the presented concept must be readable and writeable by SW applications. It should be possible for a designer to access and modify directly the content of the files, even though this may not be the main approach, because of integrity issues and eminent risk of errors. The preferred option is an application that can be used to view and modify the content of the files within the concept, thus ensuring the data integrity and the validity of the data structures.
- **Information can be carried with the module**. The information associated with the module (e.g. the description files) should be physically included within the module, while the module type and higher level information must be made available via the Internet. The basic set of information must always be available when an unidentified module is found from storage. Same applies to the accumulated module instance specific information. But then the abstracted or commonly shared information is better to be distributed over the Internet.

- **Use of International System of Units (SI)**. The SI-units [60] are used to exchange, store, and process information for the sake of consistency, clarity, and error prevention. However, this does not exclude the User Interface (UI) of an application from showing values in a localised format which is more user-friendly and comprehensible to the user. For example, bar or 'pound per square inch' (psi) can be used to show pressure instead of pascal (Pa) on the application's screen. The input, output, and visualisation of the values are separated from the formalised data. This approach is largely in accordance with the principle of the Model-View-Controller (MVC) pattern [39]. In this Description model, the data is stored as SI-units (model part), but can be viewed as another unit if needed.
- **Power and Fit of Expression**. The concept and its associated files need to be powerful enough and capable of expressing all the information needed for the different use cases defined in the framework's vision (See Ch. 5.5), such as the design, deployment and operation of the production system.
- The Description language should contain optimised versions (description layers) for serving different use-cases for the main users (stakeholders). However, the number of these layers should be minimised, and they should contain similar structures as far as possible.

4.2.3 Users

The main user roles and stakeholders for the developed concept, which were identified from various workshop materials, are presented below. These should be regarded more from the role's point of view than from the stakeholder's point of view because, for example, a *System Integrator* is often a *System designer*, or the *End User* could be the *Module Owner*. In other words, a single stakeholder could simultaneously represent several of the roles. These different roles are used in order to be able to split the requirements into as focused and isolated sets as possible, thus enabling the URs to be defined as thoroughly as possible.

1. The **Module Provider** is the supplier of the production modules for a manufacturing system, which can be either HW and/or SW. The Module Provider may provide standard modules or specialised, customised ones. In addition, the Module Provider carries out the unit tests.

2. The **System Designers** design the production system according to the requirements for the desired product. They transform the product specification into an actual production system by selecting production modules, and creating an initial production layout and production recipe. They may also perform the process analysis of the product.

3. The **System Integrator** implements the layout plan and integrates the modules together. System Integrators execute the initial parametrisation and ramp-up for the system, and they perform integration and interoperability tests. They perform the fine tuning and reconfigure the production system when needed.

4. The **End User** is the orderer and end customer of the system. They define the product specifications and the user requirements for the system. Specific sub-roles are:
   a) **Product owner**; a role that is responsible for the produced item and its specification.
   b) **End product designer**; the designer of the produced item.
   c) **System operator**; the person that operates the production system and environment.
   d) **Maintenance staff**, who maintain the production system.

5. The **Module Owner** owns the HW modules. The usage of the modules may be according to the traditional ownership model, rental business, or any sort of (new) business model.

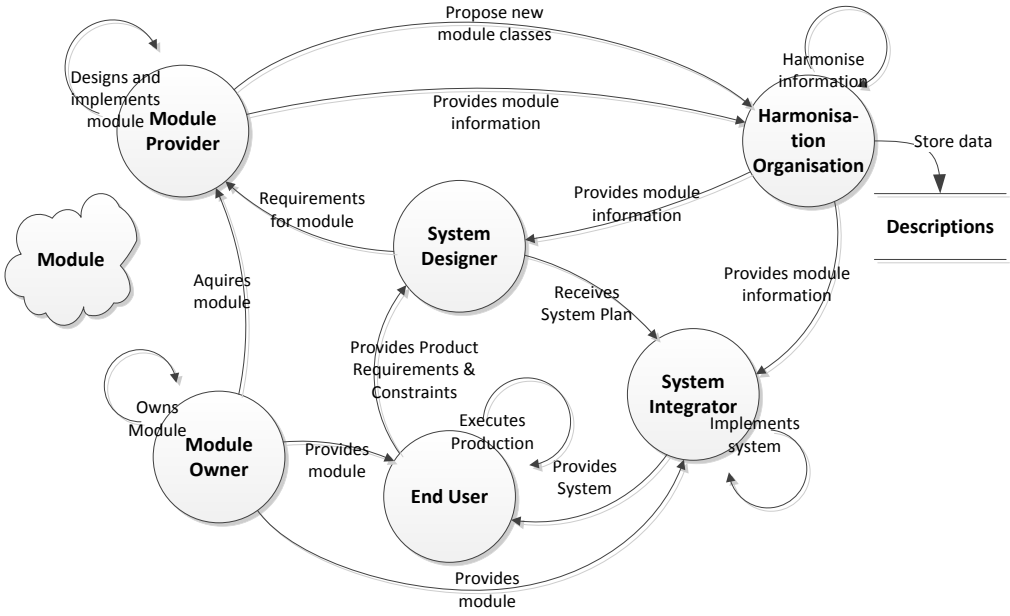


Figure 4.1: The identified users and a simplified overview of their mutual interactions

6. The **Harmonisation Organisation (or User Group)** acts as a broker between parties. It publishes specifications and descriptions, maintains consistency, and provides harmonisation or standardisation of the module descriptions. It acts as a distribution channel for information by hosting services for sharing information and knowledge.

   Harmonisation is essential to standardise the interfaces, processes, and architectures. Information for standardisation comes from other identified users (the *Module Provider*, the *System Designer*, the *Integrator*, and the *End User*).

Figure 4.1 illustrates the users involved in the concept and gives a simplified view of the interactions between them. The following sections provide more details about each user, the associated use-cases and their specific requirements.

4.2.4 The Use Cases and User-specific Requirements

The Use-Cases and User Stories are used to demonstrate the intended uses for the developed concept. The roles of the stakeholders in relation to how they can utilise the concept are explained in more detail below with the User Stories and Use-Cases.

Figure 4.2 illustrates the relationships between different users, and their involvement in the system design process. It shows the main process flow for selecting production modules for a production system, starting with the User Requirements Specification (URS). The following sections collect together different use-case scenarios associated with each stakeholder. The chapter concludes by examining the main User Requirements (URs) falling within the scope of this thesis. An extensive list of role-specific URs are shown in Appendix B, each of which has been assigned its own unique ID label.

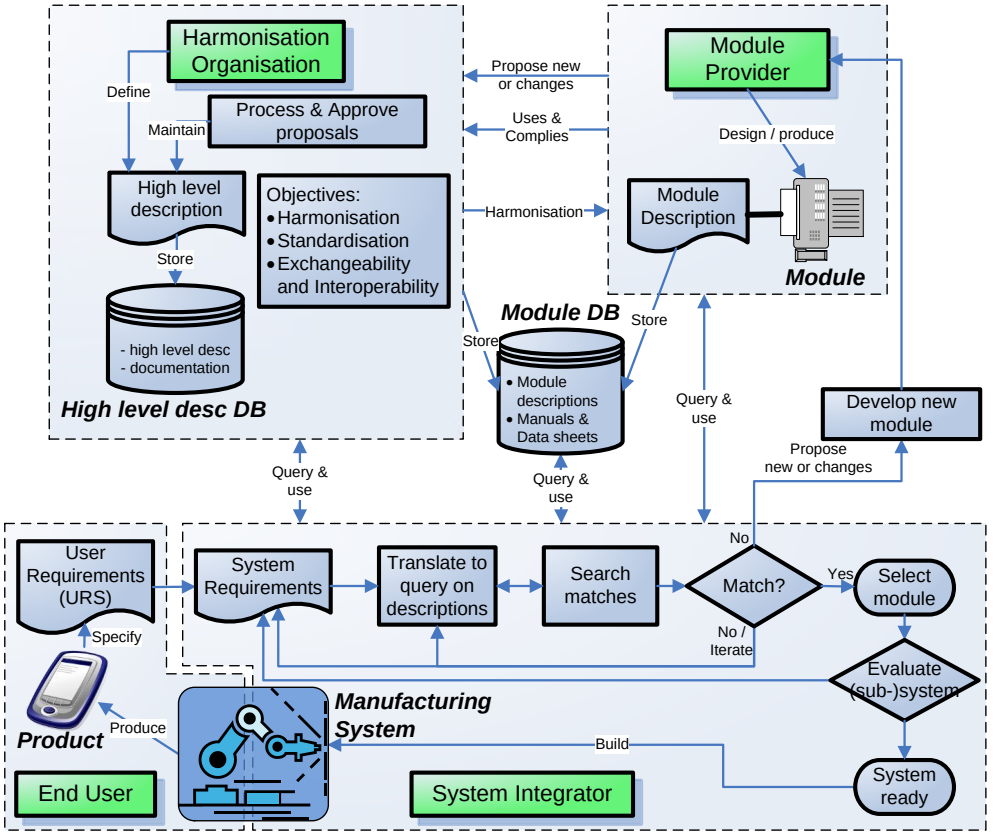


Figure 4.2: General use case and associated stakeholders

4.2.4.1 The Module Provider's perspective

The most important attribute for the production module is that it is capable of performing a necessary or desired function, process, or transformation for the produced item under production, and thus add value to that item. This could be, for example, in the processing of raw materials, in manufacturing or assembly operations, logistics, for quality assurance, or indeed almost any kind of production operation. These will be referred to as its *capabilities*, or its *behaviour*. The concept of capability has added importance as it encompasses the link between the product design (the need) and the actual production system (the output). It thus provides a practical key for opening the 'lock associated to System design process' (See Ch. 3.2.1) and the triplet of Product-Process-Resource (See Ch. 3.2.2).

Three different factors need to be taken into account when defining the information associated with the capabilities. It is important to find out the lowest common nominator for these, so that while all three are fully defined and satisfied, they are not over-defined. These factors are:

1. The production system design process must have sufficient information available to meet the needs of whatever production process sequence is required for the manufactured product. This is conducted from the Product Requirements Document.
2. The production processes or services (capabilities) can be searched for, compared, and then selected. The system can be created, integrated, and orchestrated according to the available information.
3. During the commissioning and deployment stages, the atomic capabilities are turned into

controls and control interfaces. The capability definition can be directly transformed into a control interface, which can be created or generated, activated, and finally deployed to different control architectures and production systems. These control interfaces are then used for composing or orchestrating the services provided by the modules for the working system. Module controls are connected with one another, or to the master controller, which is located at the higher level of Computer Integrated Manufacturing (CIM) pyramid.

The *Module Provider* designs and implements the process modules according to the specified requirements. These specification may be from internal sources, or external sources such as the *System Integrator* or *End User*. The specification is needed to select the interfaces, properties, and capabilities to be followed in the module design. These should all ensure that the modules can be interconnected to the system and perform the expected actions. Clear module boundaries facilitate the (unit) testing and the re-use of the modules, which in turn leads to increased quality and sustainability for the modules. Once the design is complete and the module has been implemented, the *Module Provider* produces a formalised module description which will digitally represent the physical module, showing all of its features, values, and design choices. This description is then published and made available for use by the other parties.

Among *Module Providers*, it is the SMEs who stand to gain most from the developed concept, as the concept aims to produce self-contained and easily-integrated modules. This allows an SME to focus on one specialised and innovative production process, create a module around it, and still be able to bring it to market easily. This is because the concept ensures that the module must be easily integrated into an overall production system, and can function as a part of it. The concept enables the use of partner networks in the system creation. Thus, SMEs, including both *Module Providers* and *System Integrators*, can together establish a network where each company has its own role and process speciality. Using the shared concept and architecture as the foundations of a system enable fast and reliable system integration with clear responsibilities in regard to the delivery boundaries.

4.2.4.2 The System Designer's perspective

The *System Designer* and the *System Integrator* are potentially the ones who will benefit most from the improvements offered by this novel concept and method for designing systems. The aim is to design and configure a production system from pre-defined and existing modules, instead of having to use the current, 'prototype' kind of system design process. The concept must support a module-based system design process. It should provide sufficient information to aid in comparing and selecting the modules, and to simulate and verify the production system, including the production capacity, the Key Performance Indicators (KPIs), and other key characteristics summarised under the terms capabilities, space utilisation, and collision detection.

The modular production system design process, described above in Ch. 3.2.1, where the production system is composed from ready modules (design-by-re-use), is taken as the basis of the development. The system design process is assumed to involve the following phases: product requirement identification, conceptual design (including architecture and selection of the main interfaces) and detailed design (including selection of modules), deployment, and commission [100]. The detailed design is further assumed to be an iterative process, starting with the core production process phases and selecting the modules for those first. The detailed design is then extended to the supporting and auxiliary modules, which either integrate the various parts of the production system together or support the main production process.

The trinity of product, process, and resource [65, 77, 106] are observed throughout the development. On the one hand, the resources provide for the implementation of the process(es) as these resources possess the capabilities or skills. On the other hand, the product, can be regarded in terms of the sequence of processes needed to manufacture and assemble it. Thus, the process

provides a link between the product requirements and the resources needed to realise them. The production system design process is regarded as the basic premise for the user requirements, the framework of the production system design, and the proposed concept.

The *System Designer* starts with information about the specification of the product (the URS) to be manufactured. The URS is preferably given in a formal format as the output of a tool. The *system designer*'s task is to design a manufacturing system capable of producing the required items. The design has to utilise existing modules, and is limited by the customer's (i.e. *End User*'s or *System Integrator*'s) selected architecture and interfaces. An on-line tool is used to browse and select the available modules. The designer does not need to jump from one place to another to search for the module data as a production resource pool is provided in a centralised location. Alternatively, a linking service can be provided to connect a number of resource pools together. The information is provided in a formalised, self-descriptive, and machine-readable form, which enables the information to be processed effectively and efficiently. Even more importantly, the information is directly available in a comparable format so that the features of various modules can be easily analysed, compared, and evaluated.

The production module descriptions provide information about the properties of a module from various perspectives, such as business, the environmental conditions (operation, storage, transport), its performance, quality (accuracy, Mean Time Between Failure (MTBF)), the work envelope and visualisation (Computer Aided Design (CAD) model), and for all available interfaces from various angles, such as mechanical, electrical, and communicative. The logical behaviour and capabilities (what the module can do) are also expressed formally. A tool with various filtering options can be used to find the most suitable modules for a given purpose. An advanced tool may even have algorithms for automatic selection procedures to create complete proposals for layout configurations. In this case, alternative layout configurations can be selected and compared for simulation, analysis, and evaluation purposes.

The concept should support an iterative design approach, so that the designer can start with a higher level of abstraction and then go into more detail about the design, step by step. Thus, the production system design evolves gradually as a result of the actions of the system designer, who begins with rather abstract terms such as functions, capabilities and interfaces, and then goes on to select the key processes and/or modules and place them where required. Once the basic shape of the system has been worked out, more modules or sub-assemblies of modules are appended to the system around the initial structure. By first using abstractions, and then selecting the practical realisations for them, the designer may freely move back and forth in a landscape of abstract versus detailed system design. In order to offer this advantageous feature for the designer, there must be a compatibility layer. The *System Designer* must be able to access alternative options and implementations of the available modules, preferably from another *Module Provider*. It must be possible to exchange these modules easily so that the *System Designer* can evaluate the characteristics of alternative options. The mutually compliant interfaces and capabilities, set against the abstracted requirements, are the key to the production module exchange. Any variations in the module implementations, such as different sizes, shapes or even technologies can be ignored as the system adapts to these changes and still provides the required functions. Doing this in the digital domain is not a new trick, but the novelty of this concept lies in having the same ease of exchangeability in a real-world system.

Alternatively, the entire design effort can start from an existing layout. Reconfiguration analysis and re-design can be done, and a new reconfigured layout can be found [65]. Of course, the best case would be if the physical configuration needs no changes at all. In the case of a pre-existing system, the usage history of the exiting layout and reusable modules are available a priori. The information from working configurations (from both successfully and unsuccessfully paired modules) is useful. Even if this happens at the module instance level, reasonable conclusions can be drawn at module type level. The usage history of a module can be utilised to analyse the

changed tolerances, working conditions and wear of the used modules. With this analysis, the quality of a module can be assessed and its capability to produce the requested product at the required quality can be determined. The capability definition can be adjusted by the instance description.

4.2.4.3 The System Integrator's perspective

The proposed concept should ease integration and enable a true, multi-vendor, off-the-shelf environment. The production system should be easier and faster to assemble, integrate, and set-up, even if the modules are from different origins, as the modules have to be independent and self-contained. This means that the modules contain all they need for operation without any external input. They include the mechanical embodiment, the process and functions, and control and communication capabilities. Configuring the system from existing, independent, and self-contained modules should improve the quality of the system because the modules can be tested and verified independently with unit tests before they are integrated into the system. All these factors are aimed at the system integrator being able to freely select the best-in-class process modules for their system, while being sure that everything will integrate and come together smoothly.

The smaller and finer the modules, the more efficiently the systems can be produced. The ultimate objective is to move towards systems with ever finer granularity. Traditional production lines are composed of cells, but this method goes beyond that by providing modularity inside the cells. A further objective is to compose systems from a larger number of modules of a similar type. For example, the axis system of multiple Degrees of Freedom (DOF) can be created from a set of simpler modules, each providing only one DOF. This is expected to bring scale of volume and more freedom to system design, improve re-use, and fulfil the other demands from the RMS and EPS.

4.2.4.4 The End User's perspective

The *End Users* expect to achieve improved Time to Market (TTM) and Time to Volume (TTV) for their end products (produced items). The overall manufacturing costs should be reduced compared to the existing system. The proposed solution must provide sustainability and increased quality, availability and reliability. The system must be adaptable (or at least reactive) to changes in production. This includes factors such as scalability and increase or decrease in capacity in the finer steps; faster response to faults and breakdowns (i.e. minimising down-time); and the use of intelligent and adaptive modules.

Sustainability and the security of investment are important features for the *End User*. The overall concept, modularity and finer granularity, should keep the modules up-to-date longer and reduce the risk of them becoming obsolete. An important aspect of this is the production module descriptions, which offer a powerful tool to search and configure modules for re-use. This way, any investment can be expected to remain viable for a longer period of time, even if the production is changed. From an investment perspective, this means that longer redemption times are acceptable, or the resale value of a module may be higher at the end of the amortisation period. The utility value is expected to be significantly larger than it is in present installations. More detailed knowledge about the production modules, finer granularity, and better module inventory control, combined with novel ways to design systems, will revolutionise the present culture of scraping together a whole production line when a new product is introduced. At present, businesses have no real options when designing a production system, because: a) present production systems are highly intertwined and integrated at all levels of the system - no modularity, no clear interfaces, and wires and pipes that are cut to perfectly fitting lengths, are all to create

issues for efficient re-use; b) businesses often do not know exactly what they have, often because of heterogeneous documentation; c) the operational capability of the components, the modules and the system is unknown, and d) the historical data of the resources is unknown. All of these problems would be solved with the proposed concept.

4.2.4.5 The Module Owner's perspective

The *Module Owners* have similar financial issues to the *End Users* and the new concept offers them similar advantages. These include reducing the risks involved in ownership as the modules should remain viable and functional for as long as possible, thus postponing the date of obsolescence. The ownership risk is further reduced by offering more opportunities for module usage spots and re-use after decommissioning. There will be new marketing channels for available modules (databases, broker services, and module libraries). The concept will enable new business models, such as a rental business for production modules (warehouse of modules), or charging per piece for any produced parts.

This concept will secure investments by keeping the modules up-to-date and viable for longer. This can be achieved by focusing on the re-usability and sustainability aspects of a system, and to build these into the concept as much as possible.

4.2.4.6 The Harmonisation Organisation's perspective

Module Providers, *Integrators*, and *End Users* need to have the possibility to make additions and changes to the module descriptions because there will be new technologies, modules for new areas, and new processes appearing on the market. In order to make sure that the specification process is performed in a controlled way, a central and neutral body is needed to maintain the resource definitions. In practise, the *Harmonisation Organisation* is the *User Group* promoting and advancing the concept and its associated technologies. Both the *End Users* and the *System Integrators* should first approach the *Module Provider* companies in order to initiate new proposals or changes to the descriptions. The module vendors would act a kind of filter for such proposals, as depicted in Figure 4.2.

The *User Group* is responsible for managing and distributing the descriptions. The *End Users* and *System Integrators* can utilise this linking service to access all the descriptions through one central location, without having to search for the information themselves from vendor-specific sources.

Thus, the *User Group* has an important and central role in harmonising and standardising the descriptions, which are the enablers for exchangeability and interoperability between the production resources. The User Group's task is to keep the content of the definitions coherent within a single definition, and also between different definitions, to resolve conflicting issues and to offer a fair and impartial processing of proposals.

4.2.4.7 Main User Requirements for the developed concept

The entire set of captured URs, including those which are only vaguely associated with this thesis, are collected together in Appendix B. For this thesis, only a few of the main URs are considered in detail. The same UR ID labels as used in the full list are used here as well.

- UR 1. The concept must enable and contain mechanism(s) for the exchangeability and interchangeability of production modules.
- UR 2. Interoperability and compatibility between production modules is improved comparing the current situation.
- UR 6. Enhanced modularity and the re-use of HW and SW components and description levels

UR 7. The descriptions are independent, autonomous, and self-contained. They can be used without resort to additional external resources.
UR 12. The concept and the descriptions shall not inhibit technological development and the appearance of new processes, interfaces, and features.
UR 13. The module description needs to be:
  a) capable of representing the properties of real-world production modules,
  b) readable and writeable by both machine and human experts,
  c) comprehensive and complete,
  d) easy to make and maintain,
  e) unambiguous and exact.
UR 14. Include definitions for
  a) interface (how modules can be connected together), and
  b) capability (what a module can do)
UR 19. The description language for the concept should contain optimised versions (layers) for serving different use-cases from the main stakeholders (users). However, the number of different layers should be minimised and they should contain as many similar data structures as possible.
UR 20. The descriptions can be verified
  a) by structure,
  b) to follow data models and their rules for restrictions and limitations, and
  c) by content (if possible)
UR 25. The descriptions should not reveal the IP of the *Module Provider*, but should support easy adoption and usage of the modules.
UR 27. The concept and descriptions should leave space for competition and differentiation.
UR 29. Any comparison of the modules should be easier and quicker than it is at present.
UR 32. The concept and descriptions must eliminate, or at least reduce, the need to re-create, re-type, and search for information
UR 60. Offer the means (tools) to process descriptions, i.e. to create, maintain, and modify them.
UR 61. Offer the means (tools) to distribute descriptions, i.e. to manage and share them.

4.3 System Requirements (for the Concepts and Architecture)

The following major system requirements have been distilled from the given set of user requirements presented in Ch. 4.2 and Appendix B. The System Requirements (SRs) are numbered and collected into the following lists, and categorised under topical headings.

This whole thesis is about creating a concept to model the production modules and a data model for those modules, and then to get these (the concept and data models) to work as an interface between various SW applications at the same time. Thus, in practice, much of the remainder of this thesis (Chapters 5 to 7) functions as SRs and SR documents for applications utilising the proposed concept and the module descriptions. Therefore, only some of the more general and high-level SRs are noted here.

Process The descriptions need to provide the following:
SR 1. The production system design process is to be developed and defined. It will determine what actions occur between the different users identified in Ch. 4.2.3. (The System Design process corresponding to this SR is presented in Ch. 5.5.)
  a) The system design utilises ready-made and predefined modules, and

b) it links the product requirements, i.e. the needs of the produced item to the capabilities/skills provided by the production modules.
c) The *System Designer* is supported with tools and automated processes varying from simple search criteria to more complex chains of automatic operations. [UR 38]

SR 2. The *Module provider* will have three alternative ways of creating new modules:
a) by following an existing template and to create a compliant implementation.
b) by going through the user forum to create a new template and then following previous step to implement it.
c) by creating a new module design and proposing a new template for it. The first of these should be the primary approach.

Data Exchange

SR 3. Information exchange between tools and systems must be possible with exported/imported files.
SR 4. The off-line use and processing of descriptions should be possible in the case of files travelling with the module. It is preferable that the descriptions should also be available on-line, even though this is not mandatory. The concept should not require on-line communication, but should be able to survive with off-line resources.
SR 5. Corrupted and/or wrongly generated files can be recognised ⇒ Syntax of the files can be validated.
SR 6. Wrongly formatted files can be recognised ⇒ Semantics of the files can be validated.
SR 7. Syntax and semantics checks can be performed by both sender and receiver.
SR 8. Syntax and semantics validation is an automatic and lightweight process

Description

SR 9. The descriptions need to:
a) be unambiguous and exact, leaving no room for different interpretations.
b) contain (but not be limited to) interfaces for all the mechatronic aspects, process capabilities, business and quality related properties, and physical characteristics and properties.

SR 10. Internationalisation
a) English is used as the default language.
b) Internationalisation is allowed in some pre-defined parts. These are mainly comments, descriptions, and documentation intended for human users. In these cases, the English version shall always be present and be the first or default given argument.

SR 11. Only SI units are used.
SR 12. Follow the philosophy 'Define only once, (re)use many'. This principle is followed to the extent that a thing is defined only once, in one place, which is then referenced in others. This will lead to ease of maintenance and consistency in definitions.
SR 13. Redundancy is only allowed in order to make the descriptions self-contained. Within the concept and descriptions, re-use and referencing should be used whenever possible.

Role Specific A few role-specific system requirements can be defined. The *User Group* needs the Description DB and the Link DB and is responsible for maintaining them. The harmonisation organisation may also host the module description database(s), but those can equally well be distributed to the web pages of the module vendor companies. The Link DB serves as a centralised entry point to easily find those available module description pools that comply with the concept.

The *End Users* can utilise this linking service to access all the module descriptions at once, without having to search for the information sources themselves by, for example, finding and visiting the vendor pages and downloading the required information. The ideal solution is comparable to the services and package system that the Linux distributions have for distributing SW libraries and applications.

4.4 Summary of Requirements

The goals and requirements for this work are summarised in the 'goal breakdown' map in Figure 4.3. This summarises the main objectives of this work (blue blocks) (Ch. 2.2) and how the associated requirements (orange and grey blocks) connect with and contribute to these goals. The main requirements are gathered from the specific requirements presented in the production paradigms (Ch. 3.3), especially the ones set by the RMS and EPS, the user requirements (Ch. 4.2), and the system requirements (Ch. 4.3). The main paths followed by this work are represented with the orange blocks. The grey blocks are beyond the scope of this work. It should be noted that in practise there are many more associations affecting the represented goals and requirements, but these are excluded from this work.

If the requirements at the bottom of the diagram in Figure 4.3 are fulfilled as a result of this work, it can be deduced that the outgoing associations are triggered and the effects bound to an association are realised. If all the incoming associations of another requirement are completed, this requirement is completely fulfilled and will trigger the outgoing associations arising from this requirement. Thus, the heuristic deduction can be continued. The same applies to the goals. If all the requirements contributing to a goal are completed and fulfilled, then that specific goal is fulfilled. There may also be partial effects, in which although the outgoing association may be weaker, the effect of the association still exists at a certain level. Therefore, if the proposed concept and the developed descriptions are able to fulfil the requirements leading to a goal, it can be assumed that the goal has been achieved.

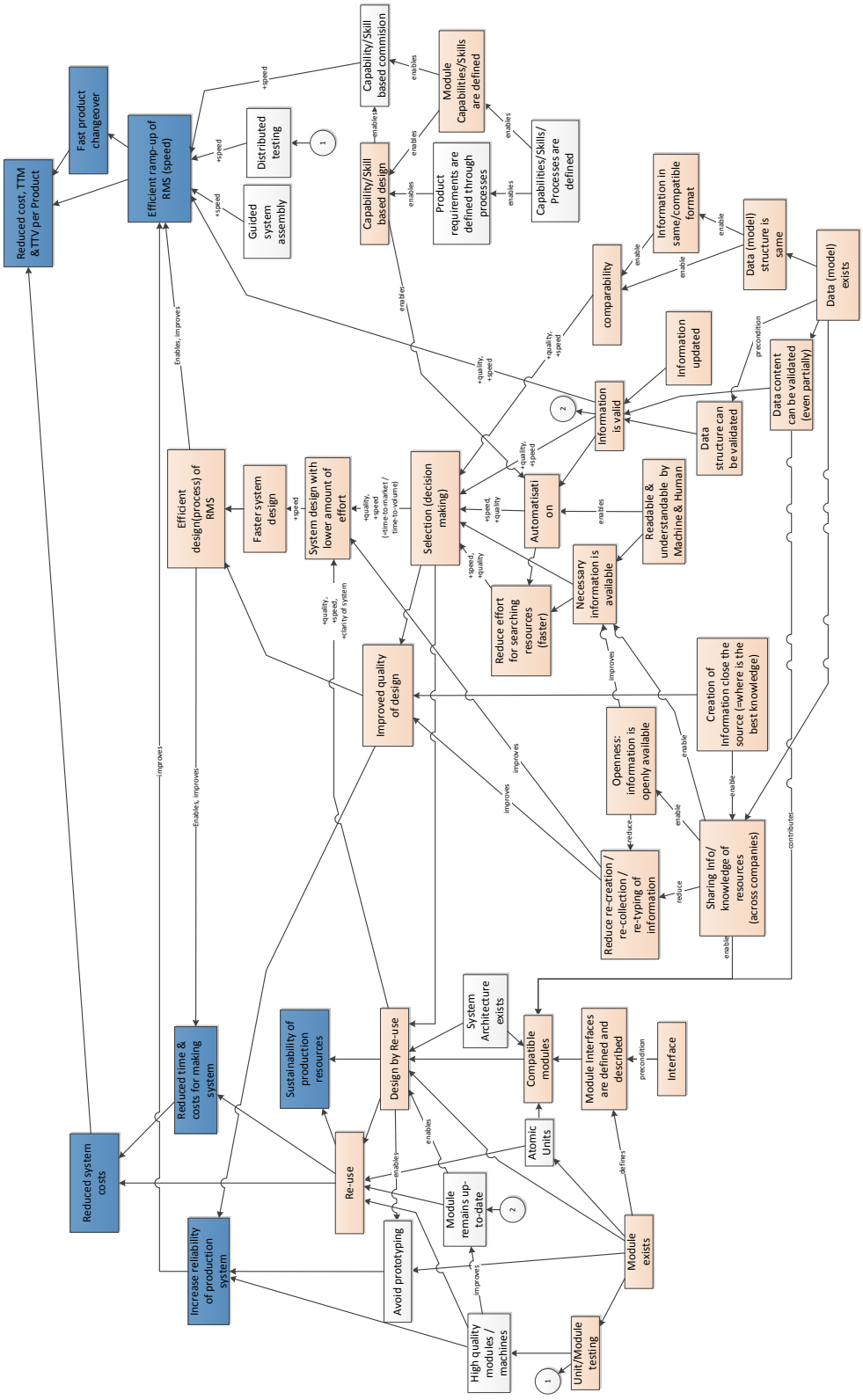


Figure 4.3: Goals and requirements breakdown map

5 Development of the Emplacement Concept and the Generic Model

This chapter defines the concept for describing the modules, and how that concept can be utilised in the design and commissioning of the production system. The chapter introduces the Generic Model for module descriptions, which can be implemented in different ways. The commonalities, common nominators and features included in the Generic Model are collected together. The idea is that these generic definitions can be applied to different kinds of implementations, but as they have the same foundations this enables semantic matching. In Ch. 6, a specific implementation is defined which gives a practical form and finish to these generic concepts, so that they can be used to describe real modules and applied and utilised with production system related SW applications.

First, Ch. 5.1 presents the definition of the Emplacement Concept and discusses the overall purpose of the descriptions, as well as identifying the number of different kinds of descriptions that are required. Second, Ch. 5.2 introduces the main entities of the Generic Model. Next, Ch. 5.3 presents details of the main entities of the Generic Model and the associated descriptions, while Ch. 5.4 summarises the Generic Model. Finally, Ch. 5.5 introduces the framework on which this thesis is based through a use-case scenario, and sketches of the system design process flow. The proposed concepts and models should fit in to this framework and support the intended scenarios and processes.

5.1 Definition of the Emplacement Concept

A definition of the Emplacement Concept is given which creates a foundation and framework for subsequent definitions and the Generic Model. The definition is:

Definition 2: The Emplacement Concept

The Emplacement Concept is the grand sum of all the descriptions representing a technical entity that integrate together aspects of production modules from the geometrical, mechanical, and functional points of view. The Emplacement Concept represents a comprehensive description of all aspects of a module's functionality, including influences from the product side, the process side, and from its interaction with other modules. It is an overarching term and encapsulates the Generic Model, which defines detailed parts of the descriptions and their interrelationships.

The word *Emplacement* is defined linguistically as [148, term: emplacement]: "1) the situation or location of something, 2) a prepared position for weapons or military equipment, or 3) a putting into position (placement)". These definitions can well be applied to our objective, which is to have

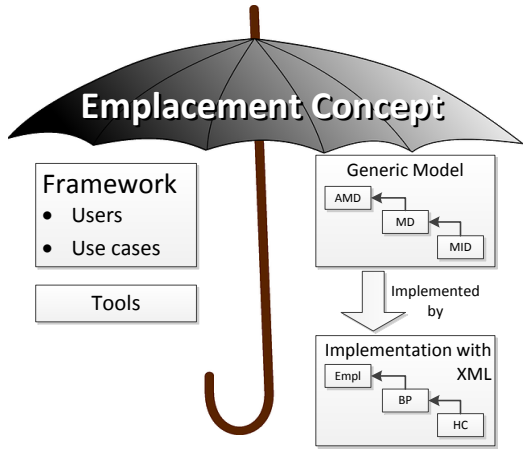


Figure 5.1: Illustration of Emplacement Concept and parts it captures

a concept or a description (See Abstract Module Description Definition 3 (p.77)) that represents a (prepared) position in a production system, into which a module can be placed and can operate as part of that system.

The Emplacement Concept is an overall concept for describing a production module. It includes a model for the production module descriptions and describes how to use them. As such, it is depicted as an umbrella in Figure 5.1, which covers a number of different elements related to the concept. It encompasses the different parts of the descriptions and defines the functions, responsibilities, and inter relationships of these descriptions. These parts and their relations are represented by the Generic Model and its internal structures. The proposed eXtensible Markup Language (XML) based implementation of the Generic Model is also included under the Emplacement Concept. However, the Emplacement Concept actually covers more than this, as it includes a framework (Ch. 5.5) for how these descriptions could be applied in practise for production system design and later deployment by the identified users (Ch. 4.2.3). It also includes the various tools associated with the concept and framework.

Figure 5.2 provides an overview of the components and their relationships within the Emplacement Concept, with the grey hatched area denoting the focus and extent of the proposed concept. These components of the concept are used to establish the Generic Model and will be elaborated on and discussed in detail in the following sections. However, before that can be done, it is necessary to determine what are the different functions needed by the Emplacement Concept descriptions, and how many parts of descriptions the concept contains.

The Purpose and Quantity of the Descriptions

As required by UR6 and UR19, the number of descriptions should be minimised and common structures and terminology should be used. A natural starting point is that a production module needs its own digital description which represents its features and properties. Module-related description can be divided into two basic sets - one that will be common and shared by all instances of a production module and another that is specific to an individual instance of a production module.

However, this is not enough to fulfill the requirements (UR1 and UR2), particularly with regard to interchangeability and interoperability, and the specific requirements of the *System Designer* (UR28 to UR38). To meet these requirements, a description at a higher level of

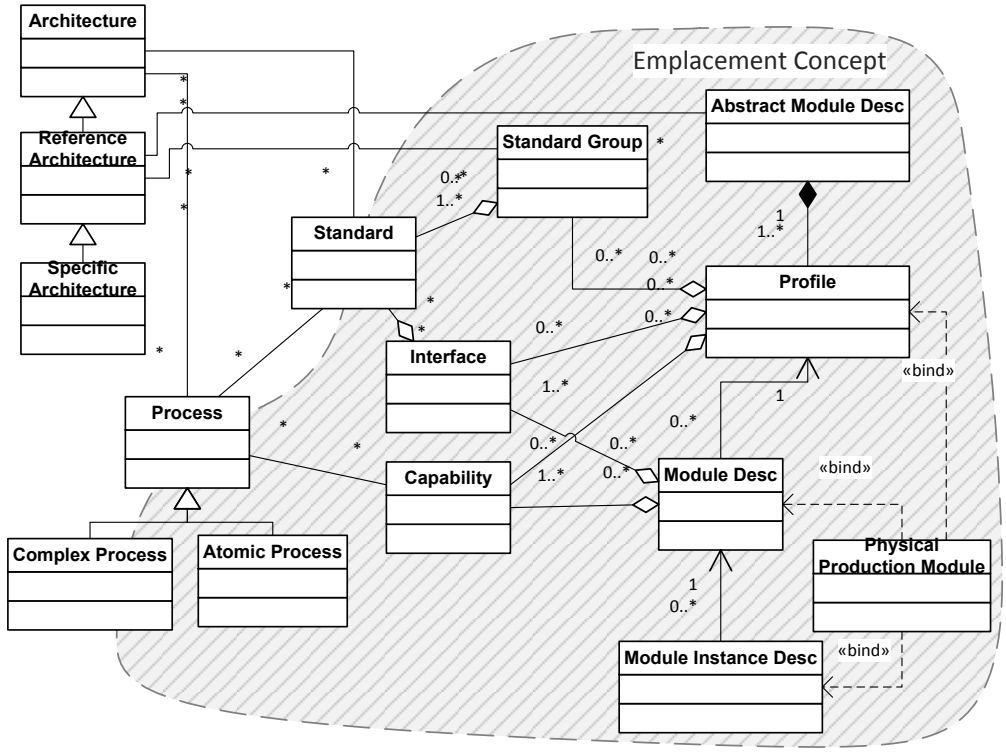


Figure 5.2: UML class diagram representing the relationship of different components within the Emplacement Concept and the Generic Model

abstraction is needed. This description should enable the planning operations to take place before the detailed design, which means that the system can be designed without locking it into a specific implementation, i.e. a specific production module from a selected supplier. This abstract level supports exchangeability because two modules can have a shared parent through which the modules are potentially interchangeable, after which the system remains (inter)operable. The features and possibilities this additional layer brings to the concept answer the requirements and needs of both the *System Designer* and the *End User*.

The Venn diagram in Figure 5.3 illustrates the identified entities of the Emplacement Concept and the relations between the Architecture, the modules, the Abstract Module Description (AMD), the Module Description (MD), and the Module Instance Description (MID). The architectural module is the physical module from which the production system can be constructed. The AMD represents the abstract high level definition that encompasses the MD definitions. Both of these are visible in the HW modules implemented according to the definition. The MID is encompassed by both the MD and the physical module as it associated with the specific instance of the production module.

On the other hand, there are modules which have no connection with, or even origins in, these definitions. This means that, a) a module has specifications, design features, and structures which are not captured by the presented concept, or b) these modules are based on different architectures than the ones used in this concept and module descriptions. Legacy modules would easily fall into this category. Thus, the number of different descriptions needed to fulfil the requirements

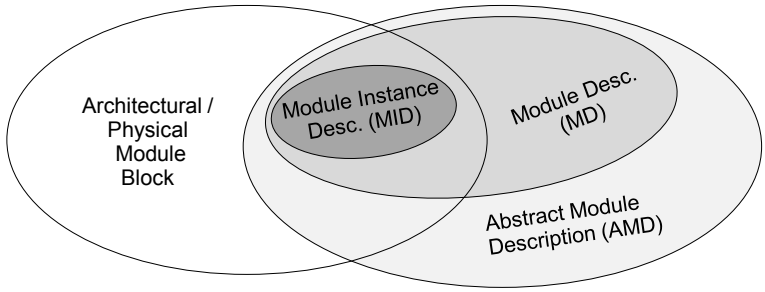


Figure 5.3: Venn diagram representing the Relationship of Architectural module, Abstract Module Description, Module Description, and Module Instance Description

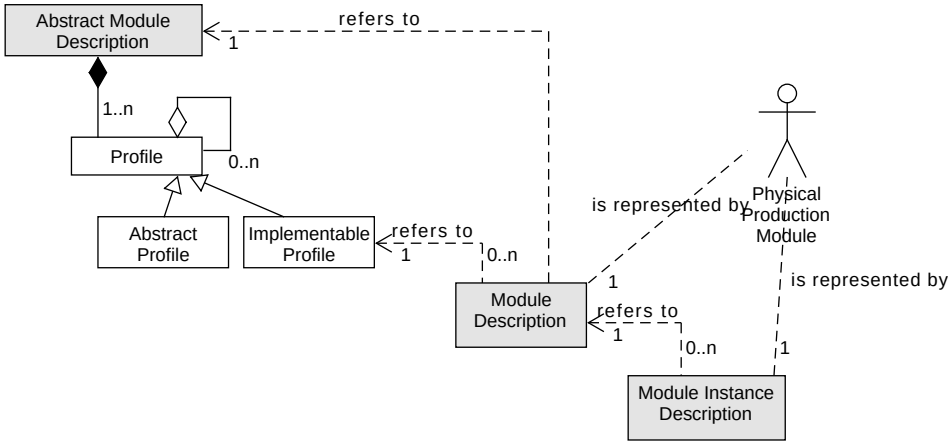


Figure 5.4: Simplified, top level class diagram of the Generic Model

is limited to these three - the AMD, MD and MID. The following sections will give detailed definitions about each of these description models.

5.2 Definition of the Generic Model and its Main Components

The following sections introduce and collect together the abstracted requirements and definitions for the models and file formats used to digitally describe the production modules in the proposed framework. This provides the foundations for the Generic Model and for the following descriptions.

Figure 5.4 represents the relationships between the Abstract Module Description, the Profile, the Module Description, and the Module Instance Description as a Universal Modeling Language (UML) class diagram. It shows how the Physical Production Module, (the actor) is associated with the Emplacement Concept. Only the core components of the Generic Model are represented in this figure, although a detailed full Generic Model is shown later in Figure 5.7.

Figure 5.5 defines the relations between the terminology used in the upcoming definitions. It presents 'Feature' and 'Characteristic' as top level terms which subsume all the rest. Directly below 'Feature' are 'Interface', 'Capability', and 'Variable Details', the latter being a common nominator

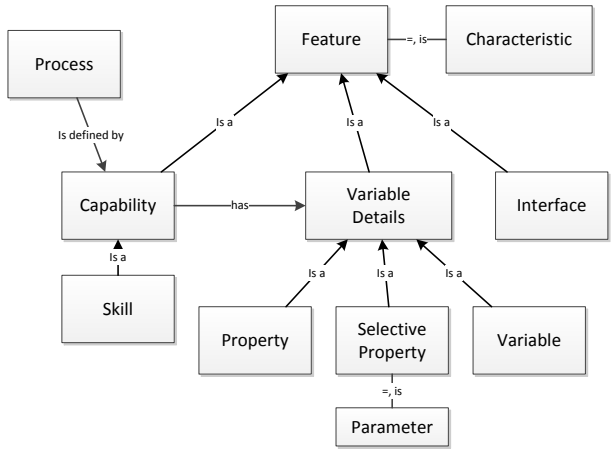


Figure 5.5: Taxonomy of terms used at descriptions

for properties, selective properties, and variables. 'Process' is defined by the capabilities. These three description entities are presented below in top-down order and defined in more detail.

5.2.1 Abstract Module Description

The Abstract Module Description is an abstract, high-level module description for a group of similar production module types. It is intended to operate at one hierarchical level of abstraction above the Module Description, for the following reasons.

Definition 3: Abstract Module Description

The Abstract Module Description (AMD) is an abstraction and a reference model for production modules. It forms a generic digital specification, defining a meta-module, an abstraction and a generalisation for grouping together similar kinds of production modules. It is a container for one or more Profiles (Definition 4), and it cannot be directly transformed into a physical module. Its task is to provide harmonisation over Module Descriptions (Definition 5) and its content is controlled by a user group(s).

The Abstract Module Description is a generic description establishing a standardised minimum set of features and requirements from which a group of production modules can be composed. It includes all the interfaces, processes (capabilities) and the number of features and properties available in those modules. These are later realised and refined by the Module Description (Definition 5).

The main objective of the Abstract Module Description is to enable interchangeability and interoperability between production modules, offering the harmonisation required by UR1 to UR3. A single Abstract Module Description can be implemented in various ways into different kinds of HW modules, but because of the inbuilt harmonisation, they are compatible and interchangeable with each other, and provide similar functionalities and features. However, it is important to leave a certain freedom of choice for the module supplier, who can utilise their design characteristics as a competitive advantage, in addition to the quality, external appearance and style of the production module.

The need for AMDs is related to the harmonisation of terms, definitions, naming conventions, interfaces, functionality and capabilities of different modules among different module vendors. The AMD works as the first step for ensuring ease of system integration, or indeed, the very possibility of doing so. Firstly, it creates the pre-conditions for interchangeability and interoperability between production modules. This is achieved by defining a description that provides a common abstraction for the modules, and represents generic information and knowledge of all the modules implemented from it. Secondly, the AMD defines the interfaces, which enable modules to be combined together into one operating system. Thirdly, it defines the capabilities which enable the modules to be linked to the process requirements for the manufactured product and its URS.

The Abstract Module Description always represents a group of modules used in a production system, never a single module. The AMD contains a generic description of the features and general requirements which are needed to integrate the module into any production system having the standards selected by a user group. The user group has a specified process to control the content of these files, so it is not up to a single provider to change the content without common agreement. Examples of Abstract Module Descriptions are gripper, manipulator, tray, and cell, which provide a digital representation of a corresponding group of modules (e.g. different kinds of grippers) which have the same general characteristics.

The Abstract Module Description specifies the standardised minimum set of requirements and abstractions which are fulfilled and specified in detail by the Module Description. These requirements cover all the common features shared by the inherited production modules, such as interface specifications, capability definitions, and the properties and variables, all of which are used to characterise the modules. The task of the AMD is to broadly define all the available features, which can then be selected (inheritance by restriction) and refined at the lower levels of description. These requirements are classified as mandatory, optional, or choice from a list. The mandatory requirements are present in every Module Description, the optional ones can be included, and in the case of choice, either zero, one, or more choices from a predefined set are present in the implementation.

The component re-use approach is also followed during the construction of the Abstract Module Description. It is composed of one or more Profiles (Definition 4), which are the main construction blocks for the description. As a matter of fact, the Abstract Module Description is a container for a number of Profiles carrying the actual definitions for specific kinds of production modules.

The AMDs and the actions of user group(s) play an important role in production module integration. These need to ensure that the mating interfaces specified in different AMDs and Profiles fit each other. Thus, the collaboration between user group(s) is very important, in ensuring that the right decisions are taken for the Abstract Module Descriptions. Fortunately, the situation is not quite that black and white, because the descriptions allow for multiple choices for a specific interface port, in addition to the concept of the grouping of interface standards (Ch. 5.3.3.2).

5.2.1.1 Profile

The main objective of the Profiles is to collect together similarities, and to assist in achieving interchangeability and interoperability between different production module implementations. They are not independent, stand-alone descriptions, but an inseparable part of an AMD. The Profile can be defined as follows:

Definition 4: Profile

The Profile defines a reusable construction block of definitions, a structure which is

used to specify the detailed section of the Abstract Module Description (Definition 3). It includes interfaces, capabilities, properties, and other features from which the abstraction of a production module is composed. The Profile is an integral and inseparable part of the AMD and cannot exist alone outside of it. Two kinds of Profiles exist, abstract and implementable ones, but only the latter can be realised as a Module Description and its physical implementation. A Profile can be built from N other Profiles using the concepts of inheritance or referencing.

The content of the Profile characterises the module, and can be used for comparison, evaluation and selection purposes. The interfaces and capability definitions specified in the Profiles are the enablers for coupling and fitting the modules together, and the composition of the production modules can thus provide the required functionality for the production task. The interfaces include comprehensive information about the mechanical, electrical, service and communication aspects and, in general capture all the information needed to connect a production module to the external world. The content and structure of the interfaces, capabilities, and the properties and variables which constitute the Profile are explained later in Ch. 5.3. As a rule, the values of properties are not fixed by the Profiles, but the profile does define the existence and purpose of a Property. The only exception is a Selective Property, for which the value is defined. This is because Selective Properties are used to characterise the Profile itself.

The Profiles are used as internal construction blocks, and for grouping specifications into re-usable blocks inside an AMD. This means that the information is defined only in one place, which can then be referenced or re-used in other parts of the description. This improves the quality and consistency of the descriptions by reducing, for instance, typing or other performance errors. It also improves the consistency of the information and facilitates the maintenance of the descriptions. The inheritance for the descriptive blocks of information fulfil the requirements needed to create the foundation for the Profile concept.

As stated above, there are two kinds of Profiles, abstract ones and implementable ones. The former are used only as internal construction blocks for the Abstract Module Descriptions. The latter are the blocks from which a real HW implementation, i.e. a physical production module, can be created, while simultaneously offering the characteristics of construction-block behaviour. The Profiles are constructed using hybrid inheritance, which includes multi-level, hierarchical, and multiple inheritances. A practical example of 're-use construction block' behaviour (i.e. define only once), and the use of these two types of Profiles is provided by a *Gripper* AMD and its Profile for a *Force-controlled 2-finger gripper*. Figure 5.6 illustrates this example, and the aforementioned characteristics of Profiles. Each rectangle is a Profile entity. The rectangles with a white background are abstract Profiles, which cannot be used to create the physical module. The ones with a grey background are implementable Profiles, from which production modules can be created. The implementable Profile of interest in this example is at the bottom of the diagram on the right and is highlighted with a thicker border. The effectiveness of the Profile concept lies in the concept of inheritance. This means that although the features of the Profile of the *Force-controlled 2-finger gripper* could all be defined from the Profile itself, instead it inherits most of its features from the seven other Profiles, represented by the connected white rectangles in Figure 5.6. It directly inherits another implementable Profile (*Simple 2-finger gripper*) and one abstract Profile (*prof.gripper.actions.finger.forceCtrl.1*). In turn, the Profile *Simple 2-finger gripper* inherits its features from four other abstract Profiles, and so on. The advantages started to accrue as other kinds of Profiles are added. For example, as Figure 5.6 shows, adding the Profile for *Position-controlled 2-finger gripper* inherits the same Profile *Simple 2-finger gripper* and its entire inheritance tree is added with the addition of only one abstract Profile containing all the features associated with the position controls. Another example would be a Profile for vacuum

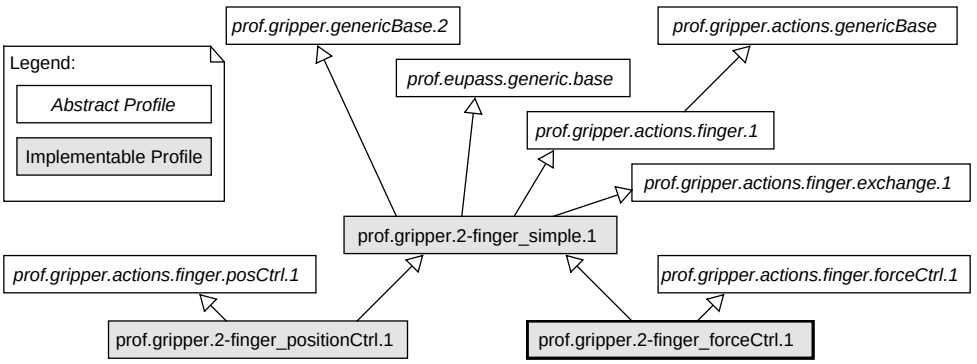


Figure 5.6: Practical example illustrating Profile inheritance

grippers, which could be implemented in parallel to the Profile *Simple 2-finger gripper* and inherit many of the same Profiles while adding a few new ones.

The implementable Profile has a clear link with the Module Description for the physical module. A template for the Module Description can be created from the Profile's information. The Module Description can also be validated against the Profile, for example by ensuring that the interface and capability instructions are followed, or that all the mandatory parts actually exist, and that the value of a certain property is within the acceptable range.

The AMD is a container, and common information and structures are shared by its Profiles, such as terms and definitions, the abstract model of the production module, common (interface) standards and links to external resources and capability descriptions. The important point is that these descriptions only need to be defined in one place, from which they can be referenced by multiple other Profiles. These features are described in detail in Ch. 5.3. The Profile contains an abstract description of alternative features associated with the production modules. These features include the references to interface standards, and the details of the capabilities and the variables. These are either introduced in the AMD's common part, or inside a Profile. With these selections, the Profile defines the requirements for implementable production modules. On the one hand, it defines the minimum set of features that must be present in the implemented module, and on the other, the set of all the selectable features or options that the implemented modules may possess. By first defining the minimum requirements, that is, the mandatory part of the description, the physical module's compliance with the Profile is assured. These mandatory features are guaranteed to be present in the production module, which means that when the Profile defines a property or interface as being mandatory, all the modules implemented from that Profile must have those specified properties and interfaces. By then defining all the selectable features that the implementation could contain, the Profile can define features (or sub-groups of features) as optional or choice according to the predefined conditions. The actual implementation thus has the option of whether or not to include a certain feature, or it can select a sub-group from a number of options. An example of choice and sub-group would be three competing interface standards, which are grouped as alternative choices containing additional and linked features. At a later stage, the module provider needs to choose one group for their implementation. In addition to this, the implementation may possess additional features which, although not derived from the Profile, can not in any way conflict with the Profile. These additional features are not present until the MD phase, so they cannot be utilised during the abstract design phases involving AMD and Profile.

5.2.1.2 Other Features of the Abstract Module Description

There is a set of various other features which are needed for the Generic Model. Each reusable or referable entity needs to have identifiers so that other entities within the model can be associated with it through referencing and cross-linking. Each top level entity, i.e. the AMD, MD, or MID, should have generic information including its name and description, the version or revision number, and the date and time of its creation. It should also include when it was last modified, contact information for the owner or maintainer and a list of the contributors. Each top level entity may also have its own special requirements, as defined below.

The AMD represents the general requirements, the abstract model of the module, the test procedures, and the module's limitations. The general requirements include an overview of the scope of the AMD. It may contain references to external definitions and descriptions, or even standards. It contains definitions of generic terms that are common to all modules inherited from this AMD, through which the abstract model is defined.

The abstract model has all the common abstractions for the physical modules. These include definitions, such as the theory of operation, abstracted interface ports, recommendations for placing the local origin, and the allowed installation positions. The theory of operation defines and models, at a generic level, what the module should do and which processes the derived modules should implement. The abstracted interface ports define the different kinds of abstract interfaces the module may have. These are associated with the different functions the derived modules have been designed to perform. The interface implementation in the Profile must refer back to the abstract interface port. Examples of abstract interface ports include the mechanical connections to the body frame, service inlets for power, communication interfaces, the imaging axis, the mobile end of the manipulator axis, or interfaces to the processed product. The concept of the abstract interface port provides additional semantics for the model, as several implementations of interfaces can be linked back to an intended function, even if the selected interface specification varies. This additional semantic dimension comes in handy when there are multiple occurrences of the same interface, but for different functions. For example, an axis or a joint module might have the same interface to connect to the body frame as it has to connect to the next link. Similarly, a communication interface can have the same interface standard for both incoming and outgoing traffic, but again the function of the interface is different. The local origin defines the origin of the module's local coordinate system, again in abstract terms. It is beneficial for the use of the module if the different production modules and module providers follow the same placement principles. The allowed installation positions and procedures impose restrictions on the use of the module. The allowed forces, including gravity and vibration, may be limited in some cases and directions. The (dis)assembly processes can also impose limitations which the model should be able to express.

The test procedures are important for the quality and consistency of the production modules. These specify the common test procedures and requirements which should be followed by all the derived modules. The procedures can be defined at both the AMD and Profile levels, depending on how generic a procedure is. The limitations and assumptions define various limiting factors for the use and operation of the derived modules. These are defined in open text formats and no specific format is followed, as they are primarily intended for humans.

5.2.2 The Module Description

The objective of the Module Description is to create a digital representation of a real production module. It defines all the interfaces, capabilities and properties with values representing the module. The Module Description is defined as follows.

Definition 5: Module Description

The Module Description (MD) is a digital representation of a real, physical production module. It gives a comprehensive description of all the details associated with a specific type of HW module used as part of a production system. The description is jointly shared by similar kinds of modules, i.e. modules having the same vendor, model, type, and version. It must contain reference to the Abstract Module Description (Definition 3) and the Profile (Definition 4) which this module is intended to implement.

The Module Description contains identifying and generally descriptive information about the production module. It contains specifications for all the mechanical, electrical, communication and service interfaces, and their spatial locations. It specifies the functional behaviour, including the capabilities, the logical behaviour, and other parameters and properties. These other parameters and properties may include the features associated with quality and reliability, performance, test and calibration procedures, and the operational conditions for the module. The Module Description may additionally have CAD models and/or photos of the module for spatial design, simulation and visualisation purposes. It may have HMI or GUI related components for generated operator views, and anything else that is common to all such modules and is applicable to the users. Some parts of the information, such as CAD models, are given as references, links or URLs to external sources.

The AMD, the Profile and the MD utilise the same definitions and descriptions for the capabilities and variable details, especially from the internal structure point of view. The main differences between specific Profiles in the AMD and the MD lie in the degrees of freedom, and in the interfaces. The purpose of the AMD's and the Profiles is to first introduce and define all the available options for the features. The Profile is derived by the MD through restriction and refinement. The MD makes selections from the available feature options, but leaves out the reset and gives values for the selected features. These may include, for example, giving the value for a property, or setting the spatial location for one instance of an interface port. For example, the Profile may define a particular property (such as the stroke of an axis) and define details such as its identifier, name, description, purpose and unit, but it does not define its actual value. The actual value is specified later, when the module provider implements the physical production module and fixes the value according to their particular production module design. The Emplacement Concept offers the following benefits:

a) The Profile harmonises the available interfaces, capabilities, and functionalities. This will aid and guide interchangeability and interoperability at the production module level.
b) The AMD Profile harmonises the terms and their units, so that the production modules (MDs) can be easily compared, i.e. the same properties are used for the same purpose and with the same units.
c) The Profile ensures coherent documentation and harmonisation of features and sets of properties for the production modules. For example, if N MDs originate from the same Profile, all their characteristics can be easily compared, one by one.
d) The module providers have the freedom to make design choices because, for example, the actual values of a property, or the selection of a particular interface, are not fixed until the module design and MD have been completed.

Further details of the Generic Model are returned to later in Ch. 5.3, which introduces specific key concepts (interface, capability, and variable details) associated with the model, and specifies their operation logic and rules.

The MD does not need the inheritance structures presented by the Profile / AMD, as they do

not provide any advantages at the MD level; rather the opposite. The complexity of the MD can be reduced and simplified by flattening the features of the descriptions and stripping down the chains of inheritance from the Profiles. For example, interface descriptions taken from an entire inheritance network of the implemented Profile can be collected under a single node in the MD, which will contain all the interface descriptions for that MD. This will make handling, processing and using the MDs faster, easier, and more efficient.

The MD is attached to the module throughout its lifetime. It can travel as part of the physical module, so that when the physical production module is collected from a storage facility, the right MD can be accessed directly from the physical module. The MD could be stored internally in the module, in a USB flash drive, or in the controller's memory. Thus, the description can be retrieved from the USB drive or directly from the module via whichever industrial communication network (fieldbus) is used, and then be utilised by various tools for commissioning, executing and operating the module. The MD should also be available on-line, as it can be utilised for the system design process, and by various system design tools, for example.

Other Features of the Module Description

The MD should also contain vendor information, a functional description, documentation and manuals, physical properties, CAD and images, GUI information, and calibration and test procedures. The vendor information defines the provider of the production module, and includes a unique, vendor-specific ID and contact information. The functional description defines the functionalities and operating principles of the module. This information is provided with links to module documentation and manuals for operation, service, and maintenance. Condensed versions of these may also be integrated into the MD.

The physical properties define the module's mass, local origin, centre of gravity relative to the module's local origin, a bounding box, installation positions, and the power supply requirements. The local origin is precisely defined by a textual description, which should follow the specifications derived from the implemented Profile. The bounding box is the minimum cuboid the module can fit in. It is defined by the module's local origin, having minimum and maximum values in each direction. The allowable installation positions and limitations include the acceleration limits in six DOFs, along with additional textual definitions. The requirements for the power supply specify the consumption of incoming energy services, such as electricity, pneumatics, and hydraulics. The power supply description should include the characteristics of the energy input (type, AC/DC, frequency, fused), the acceptable range of voltage or pressure, the nominal and maximum energy consumption, the maximum pass through capacity, and the expected function in the case of an emergency stop. The power supply requirements are most needed when dimensioning and evaluating the power sourcing network and identifying its potential bottlenecks.

The CAD models are provided for modelling, simulations, and evaluation purposes. They can be used for determining the exact work envelope, for providing information about a collision, a reachability analysis, and for off-line programming. The CAD models are added through links to external resources. The images are used to illustrate the module, or to provide an icon for the engineering tools. They can also be used to illustrate the operating principles of the module, or to provide schematics for elements such as the electrical circuitry.

The generic and compatible GUI description and models are included in the model. They can be physically contained in it, or included by reference to an external resource. The objective is to have a common abstract model for the GUI, which can be downloaded to the controller or UI device on demand, and then interpreted or compiled in the target system [32, 42, 54, 62, 87, 107, 140, 165]. The user should then be able to interact with the device, with the UI appearing and operating in the same way regardless of the OS or platform used. These characteristics will

provide potentially great savings in programming and integration efforts, as the basic level of interaction with the device is inbuilt.

The calibration and test procedures are important for evaluating and verifying the operating conditions of the module. As these procedures are specified and have their own place holders in the MD, they already have to be taken into consideration as an integral part of the module design process. These specifications facilitate and enable the unit test and calibration procedures, and thus improve the overall quality of the production system. Any production module can be calibrated and tested separately according to the specified procedures and operating conditions for the module, which in turn can be ascertained in isolation before integration takes place. The same procedure can be followed if there are doubts about the correct operation. These characteristics all have a positive effect on ramp-up and production quality. The integration of calibration and test procedures into the MD is also advantageous because they travel together with the module and are always available.

5.2.3 The Module Instance Description

The objective of the Module Instance Description is to store and carry information about a particular instance of an actual production module. Its content is frequently updated over the lifetime of the production module, unlike the MD, which is mainly static. The Module Instance Description is defined as follows:

Definition 6: Module Instance Description

The Module Instance Description (MID) is a digital representation of one instance of a module. It carries the module's current state and historical data events because it is an accumulative storage. It furnishes the Module Description with information that cannot be generalised over all instances of the same module type, but is specific to this instance only. The MID always travels with the physical production module.

The digital and HW parts need to have a strong bond so that the digital information is always present together with the module. It possesses a unique identifier which links the digital representation with its HW counterpart and it has references to its parent Module Description. It work as an accumulative storage that is updated during the life-cycle of the module whenever the module is attached to, or detached from, the production system, or even more frequently if needed. However, it is not meant to be a real-time data storage for the module, but rather a snapshot of a single instance in time, particularly when an important event occurs to the module. A server or cloud-based solution could also exist for storing MID data. This might improve or eliminate some of the security issues, e.g. lost data or data storage, and traceability issues.

The historical information might include the hours of use within a certain time range, the ownership history, the execution of calibration and test routines and their main results, its accuracy after wear and tear, and any failures or errors. From the system design and implementation point of view, the more interesting data is that generated by success or failure when making connections with other modules.

Table 5.1 summarises the characteristics of different top level descriptions of the Generic Model, by connecting these with a condensed definition and illustrative examples of actual instances of each description. The AMD is for specific technologies such as grippers, axis-systems, or feeders, and collects all the associated Profiles together. Next, the Profile provides a generalised specification for particular types of entities, such as 2-finger grippers. These are the elements which constitute a single AMD. The MD focuses on the module provider (in this

case the vendor V_A). It gives a detailed description of its particular module (2-finger gripper from vendor V_A with type or model: T1). This description accords with the definitions made in the above-cited Profile and AMD in the first two rows of the table. Finally, when the vendor V_A has produced a physical entity such as the 2-finger gripper at type T1, it is given a serial number, SN123, in this particular piece of HW.The vendor will simultaneously create an MID and connect it to this same piece of HW.

Table 5.1: Comparison of main entities of Generic Model

Entity	**Definition**	**Example**
AMD	is container for Profiles	Grippers AMD
Profile	is description of abstract module and part of AMD.	Abstraction of a 2-finger gripper
MD	is digital representation of physical module. Derived from one Profile	2-finger gripper from vendor V_A w/ type: T1
MID	is digital representation of a specific module instance.	Type: T1 gripper w/ serial: SN123

5.3 Detailed Description of the Emplacement Concept and the Generic Model

This section provides an overview and details of the Emplacement Concept and Generic Model described above in Ch. 5.1 and Ch. 5.2. The later presented framework (Ch. 5.5) is used for determining the information needs for a particular requirement. The following sections provide details of these key concepts of the Generic Model, which are the variable details, the capabilities and the interfaces. Figure 5.7 gives an overview of the detailed Generic Model.

The detailed rules of the relationships between each descriptive entity are represented with the following equations. The following First-order-Logic definitions define the relations between the components in the proposed concept.
Definitions:

$$
\begin{aligned}
Architecture &= \text{Architecture definitions} \\
Std &= \text{Standards} \\
IF &= \text{Interface (Standards)} \\
Process &= \text{Process definitions} \\
Mod &= \text{Physical Production Modules} \\
A &= \text{Abstract level module descriptions} \\
A_{Block} &= \text{Constructional component of the abstract level description} \\
B &= \text{Detailed module descriptions} \\
C &= \text{Detailed module instance descriptions} \\
*_{Inst} &= \text{Instantiation of *-type of description}
\end{aligned}
$$

Architecture is defined through architectural definition:

$$\forall x(Architecture(x) \rightarrow ArchDefinition(x)) \tag{5.1}$$

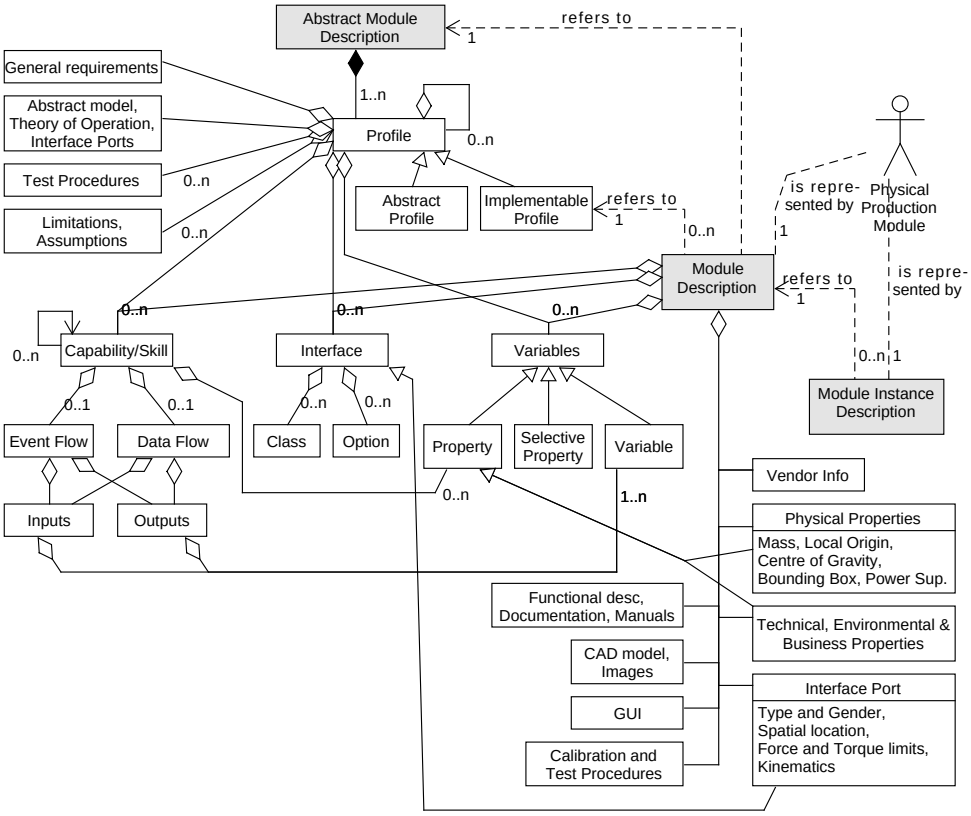


Figure 5.7: UML class diagram of the Generic Model

Architecture definition defines itself through standards, interface, or process definitions:

$$\forall x \exists y, z, w(ArchDefinition(x) \rightarrow Std(y) \vee IF(z) \vee Process(w)) \tag{5.2}$$

There exists a entity A_{Inst}, which is a constructional entity and belongs always under one abstract level module description:

$$\forall x \exists y(A_{Inst}(x) \rightarrow A(y)) \tag{5.3}$$

All Abstract level descriptions are composed from one to N reusable A_{Block} definitions:

$$\forall x \exists y(A_{Inst}(x) \rightarrow composedfrom1..NA_{Block}(y)) \tag{5.4}$$

All Abstract level, detailed level or A_{Inst} definitions relates and utilises standard, interface, and process definitions:

$$\forall x \exists y, z, w(A(x) \vee A_{Inst}(x) \vee B(x) \rightarrow Std(y) \vee IF(z) \vee Process(w)) \tag{5.5}$$

An A_{Block} is a set of Standard, interface, and Process definitions collected together:

$$\forall x \exists y, z, w(A_{Block}(x) \rightarrow Std(y) \cup IF(z) \cup Process(w)) \tag{5.6}$$

Detailed level module description is always related to a A_{Inst} description:

$$\forall x \exists y(B(x) \rightarrow A_{Inst}(y)) \tag{5.7}$$

If a module has instance level description available, it also must have a detailed level module description:

$$\forall x(C(x) \rightarrow B(x)) \tag{5.8}$$

Every implemented HW module B_{Inst} has module level description available:

$$\forall x \exists y(B_{Inst}(x) \rightarrow B(y)) \tag{5.9}$$

The module must have both level B and C descriptions in order to be compatible and compliant with the concept:

$$\exists x \in Mod(Mod(x) \wedge (B(x) \wedge C(x)) \rightarrow CompatibleMod(x)) \tag{5.10}$$

Next a few central concepts of Generic Model are defined in details. These are variable details, capabilities, and interfaces.

5.3.1 Variable Details

The concept includes different kinds of variables, which are jointly referred to as the Variable Details. Each of them has a slightly different purpose. The following entities are defined: properties, selective properties and variables. The objective is that the variable details should be defined as completely as possible within the descriptions. This includes aspects like the identifier, the name of the variable, a description, data type, unit, value (current/nominal), minimum and maximum values, increment, decade, and classification. Generally, the first five are mandatory and the latter ones are optional. The identifier and name are used to identify and distinguish the specific variables from each other. The Description (an additional purpose) defines the purpose of this variable in a human-readable form. The data type defines the type of variable according to the selected standard such as the PLCopen [52] or C -language data types. Standard SI units

are to be followed in order to make the units of measurement consistent. The increment defines the smallest resolution for the values which can be utilised by, for example, the Human Machine Interface (HMI). The decade can be used to scale the values to powers of ten and the classifications may be added in order to group properties together.

The following equations, Eq. 5.11 to Eq. 5.14, define how the data components of the variable details affect the creation of the value for the variable. Eq. 5.11 defines the total value of the variable detail. Eq. 5.12 defines the resolution for which kinds of increments the value can be changed by, which actually determines the accuracy of the value. These equations are complemented with the two additional ones (Eq. 5.13 and Eq. 5.14) which set limiting conditions to the *value* data component.

$$Value\ of\ Variable : x = < value > \times 10^{<decade>} \times < unit > \tag{5.11}$$

$$Resolution\ of\ Variable : \Delta x = < increment > \times 10^{<decade>} \times < unit > \tag{5.12}$$

If increment is set, meaning it has value different than zero, the variable value must be a integer multiplication of the increment.

$$< increment > \neq 0 \Rightarrow (\forall x)(\frac{< value >}{< increment >} = \frac{x}{\Delta x} \in \mathbb{Z} \wedge x \in \mathbb{R}) \tag{5.13}$$

If increment is not set or non-existing, the value can be any real number.

$$< increment > = 0 \Rightarrow (\forall x)(< value > = x \in \mathbb{R}) \tag{5.14}$$

Eq. 5.11 is used as common pattern for variable details. The placeholder value in Eq. 5.11 can be replaced by several aspects of a variable detail in the model, such as the current value, the nominal or default value, or the minimum and maximum values used to limit the range of allowed values for the variable detail. The purposes differ case by case. The nominal or default values give the user of the module an indication of the production module's ideal state and nominal operating conditions, and of course they provide default values when resetting the production module to its factory settings. The range defines the maximum permissible operation area for the variable. Depending on the type of variable, the range might have a slightly different meaning, varying from operating range (e.g. property: axis stroke) to maximum allowed set point (e.g. variable: process valve opening or variable: speed of axis). A more detailed level description (e.g. MD) may set a tighter range for a particular specification than the one inherited from its parent description (e.g. Profile). This is done by redefining the minimum and/or maximum values for the corresponding Property.

The advantage of all these variable detail-related definitions is that the different modules and their abstractions are easier to compare, as any set of variables is fixed by its name and its purpose. This informs the definitions and their use, as the intended purposes of the variable are written out explicitly, and they harmonise the descriptions by fixing the minimum set of variables. The variable definitions can be shared by referencing, which increases the consistency and the ease with which the variables can be stated to be the same from both a semantic and usage point of view. This is not always very clearly stated in the descriptions, which increases the difficulty of making comparisons.

Another very important aspect of the system's unambiguity is the use of standard units. This simple arrangement eliminates many areas of conflict and confusion. For example, the extra semantics around the variables make it possible to generate a user-specific GUI on the fly, as all the required information is carried within the description. Below, the three different types of variables are defined and described in detail.

Property

Definition 7: Property

A Property is a design feature of a production module, and defines a quantified value for it. The property is fully defined by the Abstract Module Description and Profile (e.g. name, unit and description), except for the value itself. The value is set later by the Module Description as it can not be set before the design of the production module has been completed. The value of a property is shared by all instances of the MD, and is constant after the design has been finished.

The properties will be utilised extensively when the production system is being designed. The production requirements, which are extracted from the product requirements, are compared and matched against the available MDs during the search for a suitable production module. In addition to the interfaces and capabilities, the properties are one of the key information sources for this match. The properties are also utilised during the simulation and verification phases for the production system. The Properties represent important business differentiators between modules, as they define the performance, quality, and other sales features of the modules. They need to be set honestly and realistically in order to correspond to reality and verified performance.

The properties can be classified under specific categories. The General Model defines such categories as physical, technical, environmental, and business properties. The physical property is a physical characteristic of the module, such as its mass or centre of gravity. The technical properties describe a technical feature or characteristic. These are used during the system design phase to select the module which best matches the product requirements . Examples of such technical properties are the stroke of a gripper, the created compression force, and the acceleration along an axis. These are also related to process-related parameters, like power and wavelength in the case of laser units, or volumetric flow for a dispensing device such as a glue dispenser. The environmental properties are used to express the environmental conditions which apply to the production module, such as acceptable temperature or humidity ranges during operation or transfer. The business properties, of course, are associated with business and production-related characteristics, such as the costs (acquisition, rental, servicing); maintenance, service, installation and calibration times, MTBF, Mean Time To Repair (MTTR); calibration intervals, the consumption of material or services during operation, and standard phase time, etc. The classification into different categories can be implemented, for example, by grouping, hierarchical organisation, inheritance, or by any other applicable method.

Selective Property

Definition 8: Selective Property

A Selective Property is a descriptive property that is used to express a technical (design) characteristic at the AMD level. It is used to differentiate the characteristics and purposes of the Profiles from each other. The same property name may be present in many different Profiles, but with different values. The value of a Selective Property is fixed at the Abstract Module Description/Profile level and will be constant for all Module Descriptions arising from it.

The Selective Properties are intended to capture common design or implementation features at the AMD and Profile levels. This information can be used to determine a suitable kind of process

module for a required production process. The concept of Selective Properties means that the search can be initiated already at the higher abstraction level, before the provider has any clue about the available real implementations of the modules. Therefore, these are especially useful during the first, conceptual, system design iterations, when the design of the overall system is still looking for its final form. This information will be in a structured format so that any application program can understand and use it, i.e. the semantics are included.

A good example of a Selective Property is the number of fingers for a Gripper AMD. The various implementable Profiles include a Selective Property called '*amount of gripping fingers*' which contains the number of fingers the gripper has for grasping an object. Thus, the different Profiles will define it differently, e.g. in the case of 2-finger gripper the value is 2, in a three-finger gripper the value is 3, and so on. This value will also define implicitly certain technical characteristics, such as how the object is centralised; and explicitly where the grasping interfaces are, and of course, the number of fingers needed. Besides all the above, the Selective Property communicates the generalised design intentions from the design of the AMD towards the module providers and their specific designs, and in addition, it harmonises this feature or functionality of all the modules implemented from the Profile.

Variable

Definition 9: Variable

The Variable is a placeholder for temporal information associated with the production module's current state or making a change to it. The value of a variable can change during the execution phase. The Abstract Module Description and the Profile define a variable completely, except for its value. In this case, the values are not set by the Module Description either. A variable is potentially a communication input or output (port) used by the control program and linking together the event or information flow between modules. Variables are also used for providing (user) controls for the module and receiving status information from the device.

Even though the AMD, Profile, and MD do not define the value of a variable, they may define a default or optimum value, and they may also set the maximum range for a specific variable when needed. When moving from one description level to another (e.g. Profile $\rightarrow$ MD), the range or default value can be refined, normally more strictly. More strictly does not necessarily mean narrowing the range, as the variable values are dependent on the physical quantity. For example, in the case of the allowed operating temperature, stricter could be the opposite to narrowing down. Examples of such variables are the state of a valve (open or closed); the current position of the fingers, an axis, or a manipulator; or the current velocity along an axis.

Three different kinds of variable details are defined in the context of the Generic Model. These are the properties, the selective properties, and the variables. Table 5.2 summarises the use of these variables and the differences between them. The main differentiator for the purpose and use of variables is the level at which the value is set.

5.3.2 Capabilities

A capability is the most important aspect of a production module. It means that module is able to perform a valuable function, process, or transformation for the produced item, and thus add value to that item. The concept of capability creates the link between the product design (needs) and the production system (output) (Ch. 3.2.1) and it completes the central part of the triplet of Product-Process-Resource (Ch. 3.2.2).

Table 5.2: Summary of variable details and their characteristics

Variable type	Value set by		
	Abstract Module Description / Profile	Module Description	Execution and use
Selective Property	X	I(1	I(1
Property	-	X	I(1
Variable	-	-	X

(1 = Value inherited from higher level description (from left)

Legend for the table: '-' not set, 'X' set, and 'I' value is inherited from level above.

The processes and functions of a production module are modelled as *capabilities* and *skills*. The former is a broader and more abstract term and contains the information needed for production system design, specifically in relation to the process sequence. This information is used for searching, comparing, and finally for selecting the production modules. The skills are a more specific representation of the control interface for a production module, enabling the creation of architecture-specific control interface implementations which can be used to generate control code in different control architectures and systems. The skills are further utilised when the controls and operations of the production system are composed and orchestrated from the services provided by the production modules, or later when the controls are finally deployed to the operational control system. The controls of a module are connected by use of skills with each other, or to the master controller at the higher level of a CIM pyramid.

Capabilities can be further classified as atomic and complex capabilities. A *complex capability* is made up of a set of atomic capabilities, or other complex capabilities. An *atomic capability* cannot be further divided, and can also be expressed in terms of skills. Specific to the skill is the existence of an event flow which differentiates it from its atomic capabilities.

The main characteristics expressed for a capability are its id, its name, a description, which are used to uniquely identify the capability; the inputs and outputs for data flow; and the properties. For a skill, there are additional characteristics regarding the inputs and outputs for event flow and how these link to specific items in the data flow. The description should give enough information about the capability for it to be selected, or rejected. When necessary, the selection can be aided with information from the data flow items.

The AMD/Profiles and MDs represent and use the capabilities in the same way, in that the structure and content of any single capability description are the same. The difference between the two is that the Profile presents a larger set of capabilities, which are offered as alternative sets (such as capabilities within an abstract Profile) or options. The MD, however, selects only those capabilities which are actually implemented in the module out of the set provided by the Profile. The following holds for the amount of available capabilities in different descriptions: $Capabilities_{AMD} \supseteq Capabilities_{Profile} \supseteq Capabilities_{MD}$. Additionally, the MD has extra fields linking the skill to a specific implementation technology. It should be remembered that the MID does not have any capabilities.

The description alone is not enough for the module selection process, for which additional, more powerful, concepts are required. A mechanism (external or inbuilt) should exist in order to link and classify the capabilities into hierarchies and process knowledge, as in, for example, [65, 73, 75]. Open taxonomies, or ontologies, for the production process domain. This could be of great use and value, but these are beyond the scope of this work. However, such organisations can be productively used during the selection process for the production modules when designing the production system. This is because the capabilities in the Generic Model and those in the

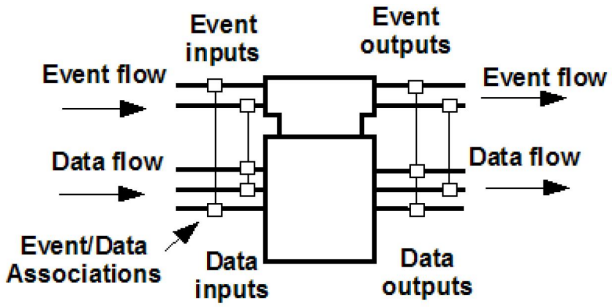


Figure 5.8: The function block type defined in IEC 61499-1 [14, Fig 1]

production modules could be linked into such an organisation. Process taxonomies or ontologies could be utilised for defining the relationship between the atomic and complex capabilities. Because the product requirements are normally defined by higher-level complex capabilities, while the production modules normally define its capabilities by skills, atomic capabilities, or at best low-level complex capabilities, such taxonomies or ontologies would be very useful in providing routes connecting these two extremes together and matching the product requirements to the offerings of the production resources.

The skills or capabilities, both atomic and complex, need to be connected together during the practical implementation of the production system. In order to accomplish this connection, the interface model IEC 61499 [14, 50] is followed at the principal level, in order to model the capabilities and skills in the Generic Model. Two set of flows, are defined for the skills - event flow and data flow, and these are further divided into input and output flows (See Figure 5.8). The first flow set is associated with the event flow of the skills. The event inputs can be things such as initialisation, triggering, enabling or resetting. Correspondingly, the event outputs can be things such as initialisation done, execution done or error active. These all are events or enablers which trigger and activate a functionality or capability within the production module. Normally these are boolean (true or false) types of variables.

The second flow set, associated with both skills and capabilities, is the data flow. It defines the variables containing the parametric information for the skill or capability, both as data input or data output. When implementing a Generic Model, there should be an association that defines which data flow-related variables are active and go with a specific event in the event flow. This is illustrated in Figure 5.8 with vertical connections between an event and items in the data flow.

The Variables (Definition 9) are used to model the data items associated with each set of inputs and outputs for both event flow and data flow. These include the identification, naming and unit information. The value of a variable is not defined in any of the description models (AMD, MD, or MID), but only later, in the operating system. However, the MD may define a default or optimum value and/or an acceptable range, as needed, as described earlier in Ch. 5.3.1.

Even though the interface model of IEC 61499 [14, 50] is adopted for describing capabilities and skills, the implementation of the Generic Model is not limited to this standard. The implementation of the controls and control interface is intended to support other control and communication standards and technologies, like Service Oriented Architecture (SOA), Web Service (WS)/Web Service Description Language (WSDL) or Device Profile for Web Service (DPWS), RESTful/RSDL, CORBA/OMG IDL, OPC Unified Architecture (OPC UA), or Programmable Logic Controllers (PLCs) [50, 52]. The objective is that other types of interface implementations can be generated on demand from the models presented by the Generic Model and the Emplacement Concept. The supported communication and control technologies and interfaces are indicated explicitly by the MD.

5.3.3 Interfaces

After the capabilities, the interfaces are the second most important feature of a production module. An interface can be defined in the following ways. [172] defines the interface from three different perspectives: a) the User interface, either with knobs and switches, or a GUI; b) a Programming interface (i.e. an Application Program Interface (API)) with statements and functions, and c) the physical and logical arrangement supporting the attachment of any device to a connector or to another device. Although the third definition is best suited in the context of this thesis, the two other are not far off the mark, either.

In the context of Emplacement Concept, two additional concepts are connected to the interface, the interface description and the interface port. When speaking generally of an interface, it is the former concept which is used. This is a specification or a standard that is formalised through the data model, because normally these are text documents meant for humans. The concept of interface port is linked to an interface description. These are the connection points at the outer edge of the production module, used to connect the modules together. The fitting conditions between these ports are searched for when the production system is designed, because only interface ports with the same set of features are connectable. The connections between the production modules and the associated rules are discussed in detail in Ch. 5.3.3.1. The interface description and the interface port are further categorised as mechanical, electrical, service, or communication interfaces, which types are used to classify the interfaces.

The interface standards normally contain various options, configurations, and selectable features. A single standard may contain a number of such definitions. Traditionally, these are either written as text or described with a graphic model in the standard document. Therefore, they need to be simplified and formalised, so that they can be modelled and processed by programs, especially the ones making decisions about mating conditions between the production modules. The Generic Model proposes two additional entities for the interface description, *Classes* and *Options*, which could be used for these purposes and associated with the interface specification reference. Their definitions are given below.

Definition 10: Class

The Interface-Class is a list of selectable values from which the module provider chooses one for their module. The values represent a feature or aspect of an interface standard, such as a size class (e.g. 1, 2, 3) bound to a certain interface. The full list is present in the AMD / Profile. In the phase of implementation (MD), the values of the classes are mutually exclusive. Thus the module vendor may implement the interface according to only one value for the class (e.g size 3). The choice of the module vendor is indicated in the MD.

Definition 11: Option

Interface-Options are representations of optional features or optional specifications existing in the standard documents. Like the class, the option in the Profile lists all the acceptable values related to it as tokens. There can be only one value. The module provider shows their final choice by having the optional feature either absent or present in the MD at the selected value.

The interfaces are used differently for the AMD and the MD. AMD interfaces are defined at a higher level of abstraction in which certain interface specifications are followed for a specific

abstract interface port. Further, each given interface description may contain multiple optional configurations. For example, the AMD / Profile may define that, in a particular case, for a specific abstract interface port, N interface standards are accepted. With regard to an interface standard, the Profile may further define that, for example, M sizes specified at one of these standards are accepted for that interface standard. These options can be represented as a list. When implementing the physical production module, the module provider opts to use one of the N interface standards and picks one of the M sizes. This selection is indicated by deleting all the other choices for the N interfaces, and deleting all the other M options associated with the selected interface, leaving only the desired options. These selections are then incorporated in the MD. Instead of deleting unwanted choices, another way of marking the selection can be expressed through the implementation of the Generic Model.

The AMD defines all the options for the interface, but gives no specifics for interface location, whereas the MD focuses on vendor differentiation and represents all the characteristics of a real, physical production module. So, additional details of the interface and interface port need to be defined in the MD. It needs to pick the interface standards and represent the exact location of each and every interface port. Each port is linked to a specific interface description with detailed choices for classes and options. Additional mechanical limitations may be expressed, as well the kinematics of the interface port arising from the module's origin. Therefore, the description of the MD interface port also includes the type and gender-related matters, spatial location, force and torque limits, and kinematics. These are further elaborated in the next section.

5.3.3.1 Connection between Modules

After identifying that a modular system like RMS or EPS requires modules, modularity, and interfaces, an important question is raised: how can the connection between modules be formalised, and more to the point, can the two modules be connected together?

This brings us back to the interfaces and their specification. On one hand, the interface can be described in a very simple way, like a single ID string, or by a very complex representation as in a Three-dimensional (3D) CAD model. Comparing single string IDs is a very straightforward way of identifying whether the two interfaces are connectable. However, this is only half of the truth. Additional dimensions and properties like type, gender, size, or variant information are not visible, or at least cannot be semantically represented by an ID. In contrast, the CAD model represents the shape and form of the interface exactly and exhaustively, with only some functionality-related aspects missing, such as moving mechanisms for locking; electrical and fluid supplies or, logical behaviour. The drawback of the CAD model-based representation is the amount of information, features and algorithms needed to answer a simple 'fit' or 'not fit' question. The verification process needed to make this judgement becomes unnecessarily exhaustive and complicated. Therefore, something between these two extremes offers an optimum solution. The solution must be lightweight, but provide more semantics and precision than a simple ID through the addition of formalised information about the interface.

Such an interface model is proposed here, where the interfaces hold a few descriptive characteristics, which specify the connectability and are common to all the different interfaces. The selected characteristics are: category, type, gender, classifiers, and variants. These are described in detail below.

- The **category** defines the intended high-level purpose of the interface, such as mechanical, electrical, service, or communication.
- The **type** must not only define the type of interface, but must also provide a link to the interface specification.
- **Gender** defines the specific mating features for the interface. There are three main options for the gender: Male, Female and Neutral. As generic examples, the Male is the pins side of

the plug, the Female is the socket, and in a neutral interface, there is no difference between the sides, but all implementations are the same from the perspective of the mechanical connection. The rules between these three can be defined as follows. Male and Female of the same interface can be united. Given certain conditions, the Male and Female interface of different standards may also be connected. Two Males or two Females from the same interface cannot be connected, but two Neutrals can be connected together. A Neutral interface may also be connected together with a Male(s) or Female(s) under certain conditions.

- The interface standards usually specify classifiers as containing many values. Examples of such classifiers are size, accuracy or communication channel. These are intended to meet the different requirements set by the various utilising applications. Even if the size is different, they all follow the same standard specification and thus belong to the same standard interface. The concept of **class** is intended to help here.
- The interface often has a *variant*. Variant means that there is a some kind of physical or logical difference or change in the implementation of the interface. The variants are different features, such as the electrical or fluid supply; different communication media or protocols, and a variant of the mechanical features on the interface. The concepts of **class** or **option** are intended to help here.

The above list provides finer granularity for the interface model, which can be utilised when deciding whether two interfaces can be mated. The interface model can also handle matrix-type interfaces, such as an optical table or a Lego block. These repeat the interface a modulo in one to three dimensions. In this case, several modules are connected to a matrix-type base interface and each module occupies $\mathbb{N}$ individual interface unit(s) in one to three dimensions.

As a result, the rules for connectable interfaces can be defined as follows. The two interface ports are always connectable if

1. the genders of the interfaces are male and female or two neutrals
2. the rest of the aforementioned characteristics of the interfaces, i.e. category, type, class, and option, are the same.

There may be other connectable interface pairs, but defining them needs additional knowledge which cannot be captured with the proposed model. Interface connection libraries could offer a solution. These could be implemented as a list containing pairs of connectable interfaces or using ontology-based definitions. However, a library of connectable interfaces is more closely associated with system design and system layout, which places it beyond the scope of this thesis and a topic for future work.

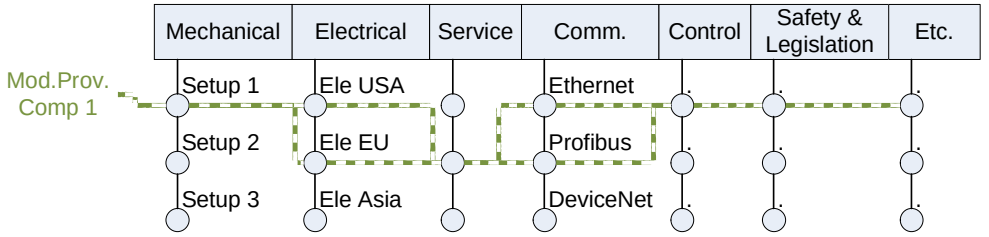
5.3.3.2 Grouping of Standards as Constructional Modules

To harmonise and ease the use of interface definitions, the standards around specific abstracted interface boundaries can be grouped together. This will provide a helpful wrapper, which can be utilised to make different modules fit better together, such as a robot and gripper.

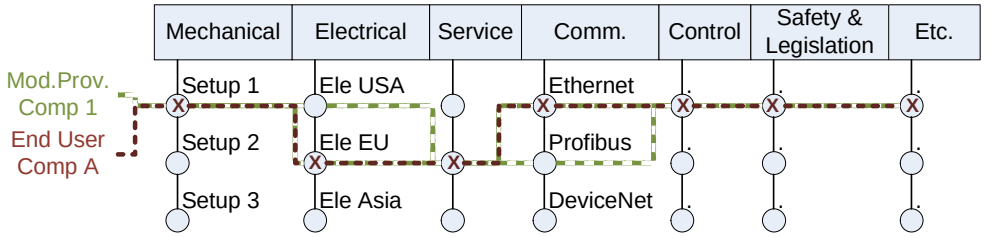
The standards for a specific area can be collected as a group of standards, which would then form a reusable block. These collections of standards are represented as circles in Figure 5.9. The figure shows the different areas in the rectangular boxes at the top of the diagram. The different collections of standards are then grouped under one area (i.e. the vertical axis, shown as a drop line). The collections may have the same standard presented in more than one collection, but this is not often the case. The different main areas are on the horizontal axis. As there should not be too many parallel areas for comparison, this proposal only lists the mechanical, electrical, service, communication, control, and safety and legislation areas. The areas, and at the same time the collections of standards, should be as free as possible from interlinking. Only in this way can

the blocks be freely selectable, thus providing self-contained construction blocks for use in the Emplacement Concept.

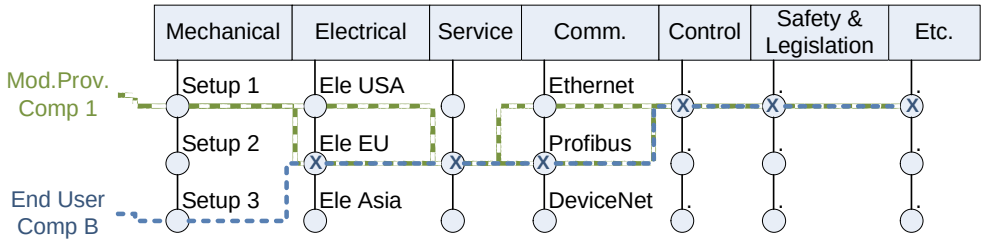
A *set of collected standards* is a horizontal pick from different groupings of standards. Each set contains exactly one (or none) collections of standards from each area. A set is, for example, the collections marked with a red 'X' in Figure 5.9b.



(a) Module Provider



(b) End User A



(c) End User B

Figure 5.9: Grouping of Standards

Figure 5.9 represents a possible scenario for the grouping and packaging of standards. The first figure, Figure 5.9a, represents the selections made by a module provider when they are implementing a device. It shows that they have selected a collection of mechanical standards, named *Setup 1*, for the implementation. The same happens for service, control, safety and legislation. In the case of electrical interfacing, the provider is following two different sets of collected specifications (i.e. grouping of standards), one for US markets (*Ele USA*) and the other for the European market (*Ele EU*). The same approach has been followed with communication. The provider is offering two different configurations for two areas, either of which the customer can select. In practice, this means that they have at least two different sales configurations available for the same device. This also means that they will have two or more Module Descriptions showing the different configurations of their device.

From the integrator, or end-user, point of view, the grouping of standards is a huge benefit. Figure 5.9b represents the case for one end-user company. They have their own preferred set of

collections of standards, which matches their architecture and facilities. Figure 5.9b shows a full match with one of the configurations from Module Provider 1. This means that the provider's module can be plugged into the system of End User A.

In the case of End User B (Figure 5.9c) this is not possible, because the Module Provider 1 does not have an implementation for the Mechanical *Setup 3*, which is requested (this is indicated by the missing blue 'X' from the Mechanical area). However, the match is quite close as all the other areas are fulfilled. Such an issue can be solved by either party changing the set of collected standards they follow, i.e. the module provider can implement *Setup 3*, or if other parts of the system allow, End User B can change their requirements to match Mechanical *Setup 1*. However, the latter is seldom possible, as the systems are usually built on a specific architecture.

5.4 Summary of the Generic Model

Figure 5.7 illustrates the summary of the detailed Generic Model as a UML class diagram, which describes the relations and associations between the defined entities. The physical production module is represented as the actor. The classes with a grey background are the main entities, defined in Ch. 5.2, and in a practical implementation they are represented as main nodes or as separate files. The associations represented here have been explained in the preceding chapters concerned with the respective entity.

5.5 Framework for Modular Production System Design

This chapter presents a system design framework on which the proposed concept and methodology are based. It shows an environment in which the developed resource descriptions are intended to operate, and shows the kind of problems they might need to solve. The requirements for the framework are defined as use-case scenarios, into which the descriptions should fit and fulfil the needs of the framework. The system design process and the related framework are prerequisites and provided 'as is'. The next section discusses the roles of the architectures associated with the framework and the proposed Emplacement Concept.

5.5.1 Design Process for creating a Production System

The systems engineering process[129], system thinking (systems-of-systems), V-model[129, Fig.1.5, Ch.5, Fig.5.3, Ch.6.4] [81], and evolutionary or incremental design model [109, Fig.7] [129, p.177–181] are taken as prerequisites for the approach followed in the production system design process. These are identified as standard engineering approaches for the production engineering domain and are therefore followed in this work. This will enable compliance and easy adoption of the proposed concept, and the description conforms to present design and engineering processes applied in industry.

The use-case presented here only focuses on the perspective of developing the proposed concept and utilises the associated files in the production system creation process from predefined production modules. The whole framework and associated processes are explained in detail below and are shown in Figure 5.10 and Figure 5.11. The labels in parenthesis within the text link the corresponding text entity to graphical blocks in these two figures.

Product Requirements The procedure for developing a new production line starts from the product design and its URS (1). The processes (i.e. capabilities) required for manufacturing the product are deduced from the product specification, and a sequence of process steps with their parameters are determined (A). The identified processes should come from standards (12),

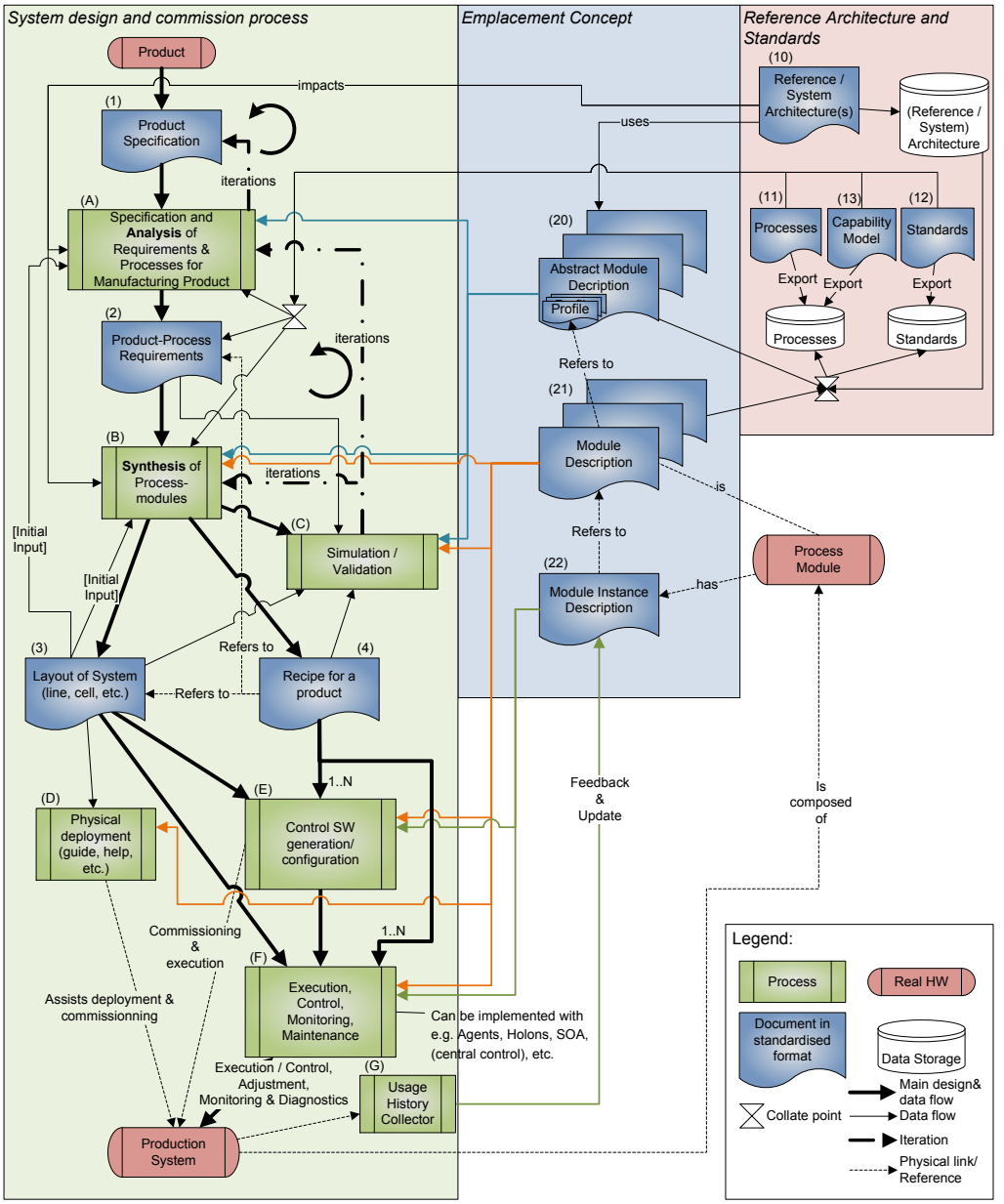


Figure 5.10: Framework used within the Emplacement Concept. Detailed components and steps used during the system design process utilising predefined production modules. The diagram illustrates the high-level design flow from manufactured product to production system. Processes (or tools) are named with letters, and documents with numbers. (Modified from [121])

(manufacturing) processes (11), and a unified process taxonomy, i.e. a capability model (13). The realisation of these sources (11 – 13) should constitute a knowledge base, which is made publicly available. It is of great importance that the agreed identifiers or names are used for identifying the capabilities and standards. Only in this way can they later be mapped with their counterparts, i.e.

the capabilities offered by the production modules. The output of this manufacturing analysis of a product specification will be the product-process requirements (2). These include the liaisons of the product components and production processes as sequences of capabilities, whose parameters build up the requirements for the manufacturing processes.

However, it must be noted that the selected capabilities describing the production process (2) are on higher levels than those of the capability model (13). This will leave some level of freedom for the subsequent stages, when the manufacturing modules and the system start to be matched with those requirements. At this later stage, it is desirable that a completely different kind of production system implementation with different performance characteristics can be created and utilised. However, all those implementations end up meeting the very same product requirements and producing products of acceptable quality.

To ensure the manufacturability of the product, the system designer can take advantage of the reference and/or system architecture(s) (10) of the end user or integrator when selecting the processes and/or standards in use. An alternative source could be an existing production system, whose set of available capabilities can be followed [65]. Furthermore, the AMDs (20) can also be utilised at this step. Information related to the specific core process can be searched out from the AMD files and the designer can start selecting preferred manufacturing processes and associated capabilities according to the boundary conditions.

Iteration loops may exist from this analysis phase (A) back to the product design and specification (1). This is necessary, for example, because of the impossibility of manufacturing specific product features, or the unavailability of the target processes and capabilities. In such cases, the iteration loops would then initiate a product change request and a new iteration.

Mapping of Product Requirements to System Offerings Once the manufacturing of the product has been split into a sequence of capabilities and processes, the next step would be mapping these capability requirements into the production modules performing those process steps (B). This step can be divided into at least to three sub-phases. Firstly, the reference architecture(s) (10) can again be taken advantage of. This may limit the interface standards used in the modules and the preferred (or mandatory) processes on the shop floor. These conditions begin to limit the amount of possible AMDs (20) [26, 173]. In the second phase, possible AMD(s) are searched out which fulfil the required processes. In the third phase, the process-module mapping is compared with the specifications of the real modules, i.e. the MDs (21) [25, 173]. At this phase, the preference could be for the modules available for use, such as the ones that already exist in the current configuration of the production system, the company's own stock or a preferred manufacturer.

Sub-systems can be built, creating larger parts of the system around the specific core process, which then creates boundary conditions. Similarly, an already available part of a line can be taken as a core, and the starting point for an iteration. One way to perform optimisation is to set a cost benefit or penalty cost associated with changes in the configuration of the production modules and system. Sticking to existing resources and current configurations may give a cost benefit, but a penalty cost would also be associated, if there is a need to break down part of the current configuration, or to acquire new production modules. Ideally, the optimisation strives to minimise system changes, thus saving time, effort and resources.

System Design Scenario – Selecting a production module The *End User* or *System Integrator* makes the URS for the product. With the help of system design tools and the information from the AMDs, the user can translate the specifications and figures from the URS into the required form, so that the tools can understand them. After that, queries can be performed and executed on the AMDs and/or MD. After retrieving the query results, the module selection can be made

from the list of suitable candidate modules, or by following an iteration loop back to any of the previous steps.

A simplified scenario for translation from URS to selected module could be:

a) URS defines: The user needs a finger gripper for grasping a part with a length of 30 mm. A 10 mm stroke is required. Open and close operation is sufficient.
b) The production requirement is formalised into its capabilities and its properties. In this case, the capability is grasping and the property is the stroke. It may be necessary to move in the capability hierarchy from combined and high-level capabilities towards simple, atomic, and implementable capabilities in order to carry out an effective search of all the production modules.
c) The above information is utilised to search for implementations providing the specified capability. The search is carried out over those AMDs which implement the requested capability, such as Gripper and Axis AMDs.
d) Looking at the Gripper AMD and finger grippers profile, the user finds out that the profile has a property called stroke. The profile also defines a set of interfaces for attaching it to the manipulator.
e) The user has already expressed a preference for the attaching interface, because he/she has already selected the manipulator for the purpose. The user can refine the search criteria: stroke $\geq$ 10 mm and the attaching interface matches the one from the manipulator.
f) The search will now focus on the physical MDs, and a list of grippers complying with the given search criteria is returned. The user can pick one from the list, or further refine the search criteria and get a reduced set to choose from. This is desirable as the resource pool for the search is large in the multi vendor, on-line market place.
g) After selection, the required verification procedures will be followed, such as, is the module available, does it fit into the available space, can the functionality and performance be verified through simulation, and are there any collisions?

During phase e) the search could also be made according to only one specific interface. As in the given scenario, criteria corresponding only to the manipulator interface could be chosen. The return value would be all the modules connectable to this spot, i.e. those which have the required interface, regardless of which AMD they belong to.

A novel aspect of this is that an option to apply the same searches to dedicated resource pools could be offered. The same search and tools can be applied to a global resource pool, to the resource pool of a specific module provider, or to a resource inventory of available modules on the specific end-customer's shop floor.

The benefits of this concept come from the abstraction level, the comparability and commensurability, and the exchangeability and interoperability. The increased level of design abstraction through functions, capabilities and interfaces leads to finding new and alternative solutions and technologies for manufacturing problems. Comparison and analysis benefit from this novel use of levelling, because the comparison can be applied not only to components and physical resources, but also at higher levels of abstraction. The use of AMDs in the middle can lead to the identification of applicable and possible configurations and combinations, even though they might not yet exist in this particular module pool. Although such modules are non-existent within the current resource pool, they might exist in others. This leads to the creation of novel system solutions. Comparability and commensurability are facilitated between different module providers, technologies and resources through the use of AMDs. This is because similar data models are used, and the information is commensurate through shared definition templates. Improvements in exchangeability and interoperability are achieved through standardisation. The module descriptions can easily be exchanged between tools and users. A large number of different kinds of production modules, originating from different processes, technologies, and module providers, can be collected together and processed with the same tools. Large resource pools can

be created, which at the same time can be shared, split, and (re-)distributed. Providing all the information in a standardised format enables easy and fluent collaboration between information producers (module providers), consumers (integrators, end users), and, in all cases, the service providers.

System Layout and Configuration During the phases of product - process analysis and process - module mapping, new "Design for X" principles can be used to create new product designs and/or system configurations. For example, the design of the existing layout, the available modules in the system or in stock, or established 'best practice' in the manufacturing processes could all affect the product design and product features. The advantages lies in using familiar and proven configurations from the past and utilising the existing line as much as possible, and only adding the missing processes to the established set-up. In this way, old and new products can even be produced in parallel on the same production line. The later re-use of the production system can be at the line level, or at any other level down to the single elementary module.

The proposed concept also enables novel operating models. The end user can utilise the AMDs and create an ideal Module Description, in the same way that module providers do when they design and implement their production modules. The (ideal) MD can be then used as a search criterion, or even as the basis for a quotation and order for a new process module if no suitable existing module is found.

As an outcome of the Synthesis of Processes and Modules (B), i.e. the mapping of modules to the required capabilities, we get the layout of the system (3) and the recipe (4). The layout of the system (3) will define which modules are connected to each other through which interface ports. The recipe (4) contains the manufacturing processing instructions for a specific product. These include the sequence of processes which a module performs on a product, along with all the parameters associated with those processes. The recipe is also linked to a specific system layout. In fact, a system layout can be linked to several product recipes, meaning that the same system configuration is capable of producing a number of different products without the need for any physical change [21, Fig.6].

Iterations As with the product - processes analysis in (A), various iteration phases take place between and within each sub-phase of system synthesis (B). It is an iterative process to determine all the possible configurations that fulfil the product requirements and capability and layout options for the system.

The feedback loop to previous steps will occur at this stage at the latest, through the simulation and validation processes (C), whose results should affect either the product - process analysis and/or the process-module mapping. The information available from both the AMDs and the MDs are used as the sources of information for system simulation and validation. The iteration through (A)→(B)→(C)→(A) or (B)→(C)→(B) may be repeated as many times as required in order to achieve a satisfactory end result. This phase should verify that the production system is capable of producing the product to the conditions and quality set by the URS and the product specification.

Deployment and Commission Once the system has been validated and proven, the decisions about reconfiguration and/or acquisition for the new system can be made. When the modules are delivered, the assisting tools are utilised to aid the operators during the deployment process (D) [174]. The AMD files, the MD files and the system layout are used as inputs. These tools can aid the system builder to put the modules together in the correct order by showing customised, step-by-step assembly instructions using interactive 3D models. Similar tools can be utilised later during maintenance operations and the disassembly phase.

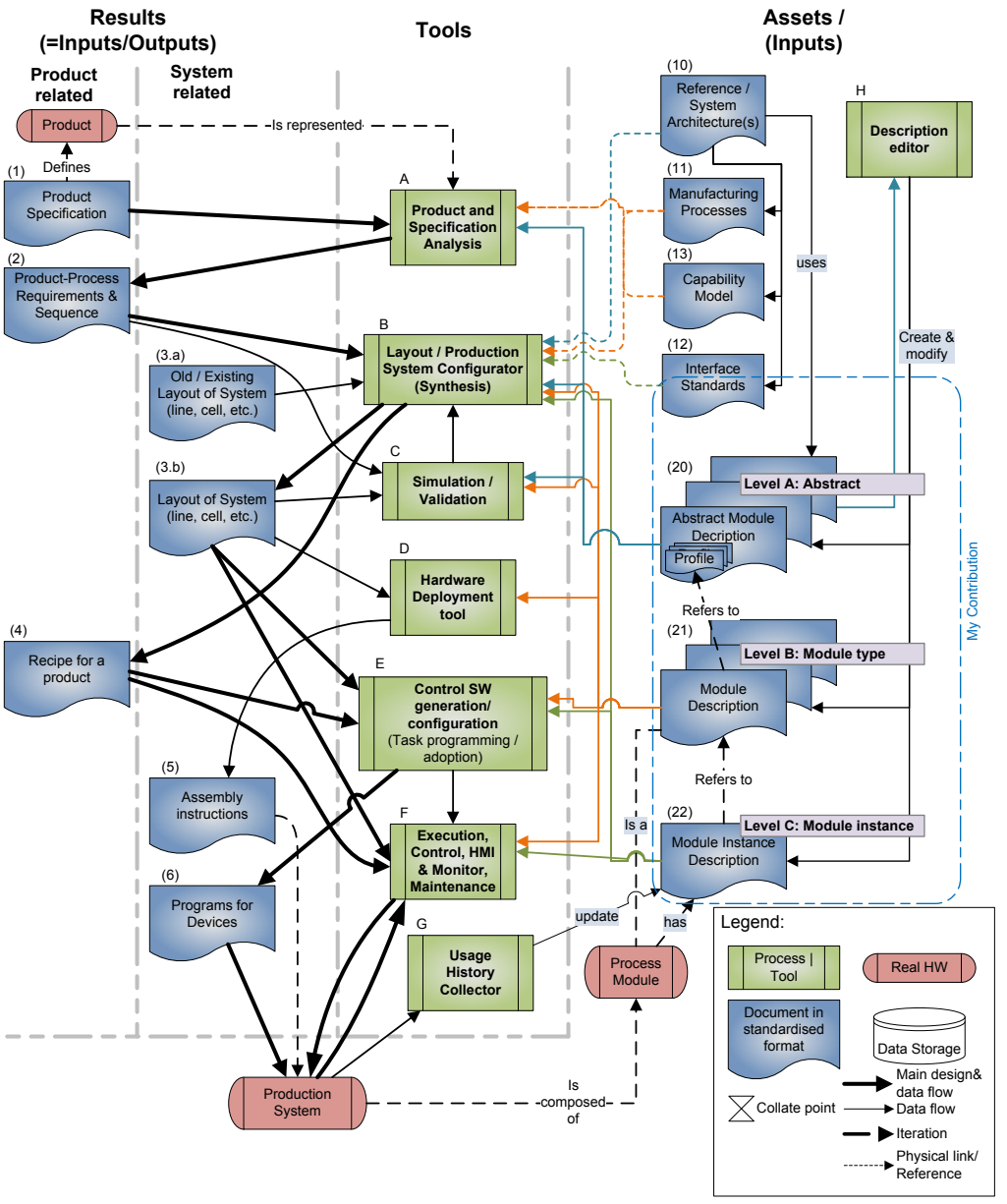


Figure 5.11: Main system design process flow for a module-based system design utilising the Emplacement Concept and the associated documents and tools.

Execution and Reconfiguration The AMD and MD files can be used in various ways during the generation or configuration of the control software (E), and during the execution of the system (F). The implementation and communication principles may vary, as well as the level of autonomous operations within the production system. This depends on both the interface standards selected for the modules and the applied reference architecture.

Figure 5.11 is an alternative view of the system design process shown in Figure 5.10. Figure 5.10 takes the assets point of view, and divides the components in the framework into three

main categories: results(Inputs/Outputs), tools and assets. The results are further classified into product- and system-related items and mainly consist of various documents. The tools category collects together the various tools which process the input documents in one way or another and produce the outputs. The assets category puts together all the assets, whether they are standards, process descriptions, production modules or their digital counterparts. The main intention is to highlight which documents are associated with the produced product, the system and to the general assets and resources.

5.5.2 Role of Architectures in the System Design Process

The framework (Ch. 5.5) and the proposed Emplacement Concept both assume the existence of an architecture (10) which acts as a foundation for the production system and its design. This architecture [16] reflects the generic decisions and selections an organisation may have already made. It explicitly states the purposes and intentions, the principles containing rules and values, the technical positions, i.e. technical guides and standards focusing on (but not limited to) interfaces, templates to be followed, and the definition of terms [111]. Architecture [16] is defined as "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution".

More specifically, the underlying architecture is expected to be a reference architecture [16, 111] or specific architecture in a domain of production systems. The former is more abstract and provides a template which is often based on the generalisation of a set of solutions. Reference architecture [111] is defined as "... an authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions.". The advantages of reference architectures are stated in [16] as being that they facilitate re-use and increase interoperability, which are, of course, also defined as objectives for this thesis. The latter is more detailed and practical in nature and is inherited from either the reference architecture or from the organisation's implicit knowledge base. The relationship between these two types of architectures may exist in several repeating layers. A specific architecture can also be the reference architecture for another specific architecture definition, as shown in [111]. Furthermore, a specific architecture may be produced from more than one reference architecture.

In the context of this thesis, the architecture in general dictates the technical solutions and the structures the organisation follows in its production systems, including the standards and interfaces that are applied. Several areas are considered concurrently, such as the mechanical, electrical, controls, and communication aspects. These, can all be defined by the same (reference) architecture, or each of them can have separate definitions originating from multiple sources. The architecture affects the production system's granularity and modularity. These aspects define at what level of the production system interchangeability takes place. The AMD can capture and support several architectures simultaneously. There can be several parallel Profiles, each of which comply with an architecture. Alternatively, a Profile can have several optional and configurable interfaces that enable a 'choice structure' for representing multiple architectures. These structures will be discussed later in more detail.

The architectures have a two-fold role within the proposed framework and concept. These roles are discontinuous over time. The first role is the (reference) architecture in the mind(s) of the *harmonisation organisation* and the involved parties (e.g. *module provider(s)* 1) during the harmonisation phase, when they design and create the AMDs and the Profile definitions as abstractions and generalisations of the production modules. In this phase, the architecture dictates the standards, interfaces, and processes followed by the descriptions. In a similar way, the module providers 2 have a (specific) architecture(s) in mind when they design and create a production module. This architecture is potentially derived from the same reference architecture utilised during the creation of the AMD by the harmonisation organisation and involved parties. The

module providers 2 match their vision and business knowledge (or in the best case, a sales order) with a corresponding AMD and Profile, and complete their module design accordingly. The second role (and what is, in effect, the third architecture) is used during the design, selection and use of the production system. The *end-user* or *integrator* has their architecture(s) to be followed in the system design. According to that definition, the AMDs, Profiles, and MDs can be judged to comply, partially comply, or not to comply with the desired architecture. The analysis focuses on interfaces and process descriptions. The first-level selection and potential system interoperability can be achieved based on this rough selection. Ideally, all three architectures would be the same and derived from the same (reference) architecture. At the very least, the definitions (which interfaces and processes to use) made from different architectures would be the same. There is a risk of obsolescence in the production modules if the module provider selects interfaces which conflict or do not belong to any end-user's architecture. This highlights the importance of common, industry-wide, shared reference architectures.

This thesis is not concerned with proposing a particular reference architecture, or any other architecture to be followed, as these issues are beyond the scope of this work. However, the existence of all architectures is taken as a given and its necessity is acknowledged in this work. The AMDs and Profiles definitions need to follow an architecture in order to be of any practical use. The different components constituting the proposed concept and their interrelationships are illustrated in Figure 5.2, which clearly shows the effect the architectures have on standards, processes, interfaces, and AMDs. Throughout these definitions and concepts, the architectures affect both the MDs and the MIDs which finally characterise the physical production module. The shaded hatched area on the diagram shows the main focus of this thesis, which is the proposed Emplacement Concept.

6 Development of Formal Description Formats

This chapter presents an implementation of the Emplacement Concept, the Generic Model, and the generic module descriptions introduced in Ch. 5. It starts with the selection of a technological platform for the implementation and the rationale behind it. Next, the implementation of a Generic Model is introduced. The focus is on the descriptions of the core concepts, namely the interfaces and capabilities. These are discussed in detail for each of the three levels of the Generic Model. Finally, general features related to the implemented description formats are discussed.

6.1 Supporting XML technologies

In order to respond to the requirements given in Ch. 4.3 (p.69), the XML together with the XML Schema Definition (XSD) and other XML family-related technologies were selected as the implementation technology for the proposed concept. These offered a natural and solid foundation for developing the concept from the interchangeability and interoperability points of view. XML provides common ground for information exchange because there are a large number of tools and program libraries which support the processing and utilisation of these files. The XML has become a "de facto" -standard for exchanging information in a heterogeneous tool and SW landscape. XSD provides a formal way to define the structure and constrain the contents of the exchanged files. Additionally it provides procedures for verifying the content of XML files. This is of great benefit for tools exchanging information and utilising the files, as they can be sure that the processed file is complete and contains all the expected information. The XML technologies are provided at a higher level of abstraction and they are more independent of tool and OS implementations than other solutions using a dedicated description language, even in binary format, or computer programs made with a specific programming language.

eXtensible Stylesheet Language Transformation (XSLT)

XSLT is a domain-specific language which is a declarative programming language. It is specified as a WWW community (W3C) Technical Recommendation (TR) [143] and it utilises other W3C specifications like XML Path Language (XPath) [146]. XSLT offers a method for document processing which can be used to make a transformation from an XML document to another document. The transformation processing instructions or rules are specified as an XSLT document, which defines how matching nodes in the input file are processed, and what is the resulting output in each case. One of the first intended uses for XSLT was to create a human-readable document out of an XML document in a Hypertext Markup Language (HTML).

The method utilises an XSLT processor which reads an XSLT document and an input file based on XML. The transformation rules are read from the XSLT file and are interpreted and applied to the input document by the XSLT processor. The rules define the content of the output

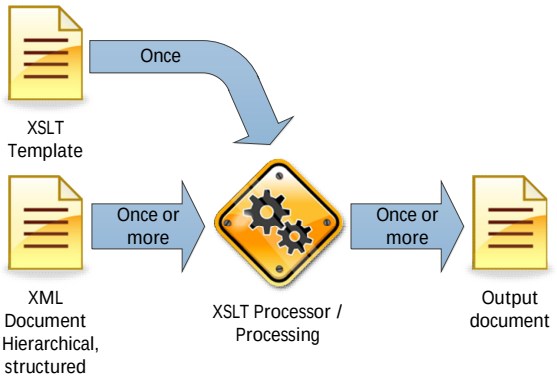


Figure 6.1: XSLT processing

generated, which could be any textual format. The rules are read in only once, and processing can be applied to several input documents in a row. The limitation is that only one input document is processed at a time, i.e. the process involves only three documents at a time: one rule set, one input and one output. After the process is finished, the next set is processed. See Figure 6.1.

6.2 Developed Description Formats

This chapter provides a reference implementation and a specification for the module description concept. The implementation is used to demonstrate the power and possibilities of the Emplacement Concept such as:

- Demonstrating the proposed concept and details of the associated files
- An implementation of the proposed methods

The following are the assignments between the abstract definitions made in Ch. 5.2 (p.76), Ch. 5.3 (p.85), and this implementation. The appointments are shown in Table 6.1.

Table 6.1: Relations between Abstract and Implementation entities

Abstract entity		Corresponding implementation
by definition	in Ch. 5.3	
Abstract Module Description (AMD)	A	Emplacement
Profile	A_{Block}	$\text{Profile}_{\text{Empl}}$
Module Description (MD)	B	Blueprint (BP)
Module Instance Description (MID)	C	History Container (HC)

The structure of the description file follows a composition hierarchy [129, pp. 75-76] so that the description goes from the root node towards the subsystems, representing the decomposition of the whole system at every hierarchical level. A hierarchical structure is well supported by the structure of an XML document, so it is useful in this context.

The implemented exchange file formats are dictated by the given specifications. The order of precedence is as follows. First, at the lowest priority, the tutorials give an overview of use. Secondly, the corresponding specification or standard defines the content and structures. Thirdly, the documentation of the XSD is generated automatically. However, the final decision and the highest priority for file formatting is defined by the XSD models, and more specifically the

selected version of the XSD description. This order of precedence will be followed in the case of any conflicts.

As stated earlier, the file formats for Emplacement [26] and Blueprint [25] are standardised by the EUPASS organisation [174]. The History Container (22) has been designed, but the format has not yet been standardised. The exchange formats for information about the standards (12) and processes (11); Product-Process requirements (2); Layout (3) and Recipe (4) documents represented in Figure 5.10 are all undefined and left for future work or external sources to complete.

6.2.1 Emplacement

The Emplacement implements the Abstract Module Description (AMD) from the Generic Model (See Ch. 5.2.1). The objective of this is to harmonise and ensure the exchangeability and interchangeability of the production modules. It assists in the system integration by specifying congruent and connectable interfaces across different module descriptions.

Definition 12: Emplacement

Emplacement is the implementation of the Abstract Module Description (AMD) (See Definition 3). It is an electronic specification of a collection of the same kind of modules, known as a meta-module. It specifies the interfaces, the behaviour and the names of available properties for the production module(s). The qualifications for the specifications are set within the document - a specification can be marked as mandatory, optional, or choice. Mandatory parts must be present; optional can be; and in the case of choice, either zero, one, or more choices are present in the implementation. The format of the file is XML-based on the given XSD, which is standardised by the user group. The modular approach is followed during the construction of the Emplacement file, as it is composed of one or more $\text{Profile}_{\text{Empl}}$s, which are its main construction blocks.

The Emplacement is a container document for grouping similar and associated descriptions together. The Emplacement file format is specified in [26]. This defines it as follows: "The Emplacement is a generic description of features and general requirements of a module for an assembly system, which are required to integrate it into such an assembly system with respect to the selected standards by the consortium. The Emplacement is a collection of interface and property specifications for a certain kind of module. It is at a certain level an abstract specification of the standards and interfaces used by a module." Examples of Emplacement are: Gripper, Manipulator, Vision unit.

6.2.2 Profile inside Emplacement

Definition 13: $\text{Profile}_{\text{Empl}}$

$\text{Profile}_{\text{Empl}}$ is an implementation of a Profile (Definition 4). It is an integral and inseparable part of the Emplacement. It defines a reusable construction block; a structure which can be utilised when specifying the content of the Emplacement. It defines the selection of characteristics, interfaces, and capabilities defining the generalised module. There are two types of $\text{Profile}_{\text{Empl}}$s – abstract and implementable ones.

Profile$_{\text{Empl}}$ is defined in [26] as: "The Profile is a technical, schematic and detailed description of a homogeneous module item for an assembly system. It defines the detailed set of features and requirements that make one module interconnected and interchangeable with another. The Profile is a specification of interfaces and properties for a module. The Profile is the representation of one kind of physical module and it can be instanced by physical hardware. Profiles further classify and categorise the emplacement instantiations inside a specific class of Emplacement. Within the profiles, the commonalities between the variants of an Emplacement class are grouped together." Examples of Profile$_{\text{Empl}}$ are: Vacuum gripper, 2-finger force feedback gripper, 3 DOF Cartesian manipulator.

A Profile$_{\text{Empl}}$ consists of a number of different specifications. These could be standards or other interface specifications, capabilities and functions, and properties. The properties are measurable features common to all such modules, but whose values will be evaluated in the implementation phase. The interface specification may be presented in detail with *Classes* and *Options*. The Profile$_{\text{Empl}}$s are the elementary building blocks of the Emplacement files. In the Emplacement, two kinds of Profile$_{\text{Empl}}$s are defined: abstract and implementable ones. The rules are as follows:

1. The module always follows the specification of one implementable Profile$_{\text{Empl}}$.
2. The abstract Profile$_{\text{Empl}}$s are used as the construction blocks, grouping together a set of re-usable specifications.
3. A Profile$_{\text{Empl}}$ can inherit zero or more other Profile$_{\text{Empl}}$s, including both abstract and implementable ones.

The multiple inheritance concept for Profiles, see Ch. 5.2.1.1, is implemented by the data model of Profile$_{\text{Empl}}$ with the concept of multiple extensions. This concept aims to increase the re-use of definitions, and the practice of making definitions only in one place and referencing them from other places. **`ExtendRef`** elements mean that entire Profile$_{\text{Empl}}$(s) can be extended (See Figure 6.2 line 20). This means that all definitions of another Profile$_{\text{Empl}}$ are inherited into this profile.

The extensive use of identifiers (`id`), and referencing them with reference identifiers (`refId`) from other entities, aims for re-use and ease of maintenance for the documents. This pattern is followed throughout the implementation, making it possible to create the definition in one place and use it in many others. However, in some cases this leads to additional complexity, as in the case of multiple inheritance of Profile$_{\text{Empl}}$s. This leads to recursion and a branching tree, which needs to be traversed, for example, in order to create a comprehensive picture of available capabilities or interfaces for a single Profile$_{\text{Empl}}$. The decision was made to favour re-use, modularity of description, and expandability.

6.2.2.1 Modelling of Interfaces in an Emplacement / Profile$_{\text{Empl}}$

An interface is modelled by three main structures. First, the *Interface implementation* (or Interface reference) is defined inside a Profile$_{\text{Empl}}$, which references the other two structures common to the entire Emplacement, the *Interface ports* and the *Interface details*. The Interface details and Interface implementations are further divided into *Classes* and *Options*. Firstly, the interface ports define abstract interface ports that specify abstracted functional interfaces in the production modules, such as connection to the previous or successor module, the imaging axis, the processing interface and the product interface. The purpose of the interface ports is to create additional semantics across the interfaces and Profile$_{\text{Empl}}$s. This way, the functional behaviour of an interface can be shared over different kinds of Profile$_{\text{Empl}}$s and, finally, the production modules belonging to the same Emplacement. For example, an abstract interface port can be used to tag the specific interface which is used for mounting the module on the preceding module. This information can be later utilised in searching for modules for alternative system configurations and designs. Interface

ports are specified in the header of the Emplacement, and any Profile$_{Empl}$s across the document can reference them. Secondly, the interface details are the formalisation of the details of the interface standards (documents). It contains the name and reference to the corresponding standard and standardisation organisation. The technical details made in the standard are formalised and opened within the description by structures of classes and options. Interface details are shared by all Profile$_{Empl}$s in a similar way to the Interface ports. Finally, the interface implementation defines which interfaces are used by each Profile$_{Empl}$, and in what kind of configuration. It references both the Interface ports and the Interface details linking them to the implementation. It gives further details of the interface implementation, like type, gender, number of occurrences and also whether the interface is optional or mandatory.

An interface model, for both details and references, contains two major sub-structures, Classes and Options. A *Class* is a list of selectable values from which the module vendor chooses one for his/her module (See Definition 10 (p.93)). It may present classes (e.g. size 1, 2, 3) for a certain interface. The Emplacement/Profile$_{Empl}$ represents all the allowed values for a class. Later, the module vendor may implement the interface according one of the given class values (e.g, size 3). The choice of the module vendor will be indicated in the BP of the module. *Options* are optional features or specifications that are either present or not in the implementation of the module (See Definition 11 (p.93)). The module vendor displays the presence or absence of the optional feature in the BP file of the module. However, in both the cases presented above, the module still belongs to the very same Emplacement and Profile$_{Empl}$, even though the implementation may look very different.

Figure 6.2 shows a simplified example of the use of the Interface concept within Emplacement. The lines 3 to 5 introduce all the abstract interface ports in the abstracted model of the device. The **`InterfacePort`** is designated an identifier (`id`), and given a `name`, `description`, and `purpose`. This port is later referenced by the given `id`, which is taken, for example, from the Interface implementations.

The **`Interface`** element defines an interface standard and its formalised description in lines 7 to 11. An attribute `id` is also set on it, which can later be used for referencing this standard, and its definition. The element is given a `name`, a `description` and a `URL` that can contain some additional information about this interface standard. Preferably, the standard document is found from the given locator. Reference link (`idRefStdBody`) is set to the ID of the corresponding standardisation body. This example defines two different classes and one option. For example, a class description on line 8 defines the mechanical size of the interface by giving the `name` and `description` for the **`Class`** and setting the interface type (`ifType`). It also defines the accepted `values` as a tokenised string list i.e. different class values are listed and separated with a space. These are the only acceptable values for this class in later stages of the implementation. Similarly, the **`Option`** on line 10 defines a fieldbus interface. The difference is that as it is defined as an option, so it may be removed from the BP file.

The standardisation bodies are introduced in lines 13 to 17. The `id`, `name` and `URL` are specified for a standardisation body (**`StdBody`**). Additionally, an eXtensible Hypertext Markup Language (XHTML) document can be associated inside the element, as line 15 illustrates. This contains free hypertext with links and additional information with formatting.

Finally, the Profile$_{Empl}$ definition specifies which interfaces are included and implemented in it. With regard to the interfaces, **`InterfaceRef`** first links the interface port and interface standards together in lines 22 to 25. For example, in line 23, the mechanical aspects of the standard (`interfaceRef` = std.ISO_29262) are linked to the abstract interface port (`interfacePortRef` = IF1) and additional definitions are made in this element. In both cases, the ids are used to reference the descriptions made in other parts of the Emplacement document. Furthermore, additional definitions are made to this interface implementation: the `ifType` categorising the type of the interface under mechanical, electrical, communication, or service interfaces; the `gender` of

```
<?xml version="1.0" encoding="UTF-8"?>
<Emplacement>
  <InterfacePorts>
    <InterfacePort id="IF1" name="Robot Interface" description="Interface of a gripper to the interfaces of robots (like manipulator or similar device) [ISO 14539:2000]" purpose="Attaching Gripper into predecessor module"/>
  </InterfacePorts>
  <Interfaces>
    <Interface id="std.ISO_29262" name="Production equipment for microsystems - Interface between grippers and handling systems" idRefStdBody="stdBody.ISO" description="End effector interface." URL="">
      <Class name="Size" values="15 20 25 30 35" ifType="MECHANICAL_PORT" description="Nominal size of the end-effector interface"/>
      <Class name="AccuracyClass" values="C M F" ifType="MECHANICAL_PORT" description="Accuracy implementation level of the mechanical interface"/>
      <Option name="Fieldbus" values="ASI CC DN ETH FW IBS PB USB" ifType="COMMUNICATION_PORT" description="(Field)bus interface designation from the standard"/>
    </Interface>
  </Interfaces>
  <StdBodies>
    <StdBody id="stdBody.ISO" name="ISO" URL="http://www.iso.org">
      <XHTMLDescription>Description of ISO organisation</XHTMLDescription>
    </StdBody>
  </StdBodies>
  <Profiles>
    <Profile moduleLevel="UNIT" id="prof.gripper.example.1" name="Example Profile with attaching gripper according ISO 29262" version="1.0" description="This is the profile defining the generic attaching interfaces and properties for grippers." isAbstract="true">
      <ExtendRef reqOpt="OPT">prof.gripper.actions.finger.1</ExtendRef>
      ...
      <InterfaceRefs>
        <InterfaceRef ifType="MECHANICAL_PORT" interfacePortRef="IF1" interfaceRef="std.ISO_29262" description="Interface attaching gripper into the manipulator. Mechanical connection." gender="MALE" reqOpt="REQ" minOcc="1" maxOcc="1"/>
        <InterfaceRef ifType="ELECTRICAL_PORT" interfacePortRef="IF1" interfaceRef="std.ISO_29262" description="Interface attaching gripper into the manipulator. Electical supply and connection." gender="MALE" reqOpt="REQ" minOcc="1" maxOcc="1"/>
      </InterfaceRefs>
      <SkillRefs>
        <SkillRef idRef="act.open.1" reqOpt="REQ"/>
        <SkillRef idRef="act.close.1" reqOpt="REQ"/>
      </SkillRefs>
      ...
    </Profile>
  </Profiles>
</Emplacement>
```

Figure 6.2: Simplified example - excerpt of XML implementation of Interfaces and its internal elements - Classes and Options

the used interface is defined; whether it is required or optional (**reqOpt**); and the minimum and maximum amount of occurrences for this interface in the implementation, in case the $\text{Profile}_{\text{Empl}}$ wants to limit the number of allowed occurrencies for this interface. The attribute reqOpt is defined in the global scope and it means the same in every case. If the value is 'REQ', the element is mandatory and must be always be present when a lower level description is made. In contrast, if the value is 'OPT', it is optional and can be missing.

SkillRefs connect the $\text{Profile}_{\text{Empl}}$ to different capabilities, in this case, skills. The skills themselves are defined in another part of the Emplacement document, and can only be referenced here. In addition, a decision is made and documented here as to whether the skill is required or optional for this profile (reqOpt). The implementation of a data model for capabilities is discussed in detail in Ch. 6.2.2.3.

In the data model, elements in plural form are used to collect together groups of similar entities, so that the **InterfacePorts** element may contain many **InterfacePort** elements, and the elements **Interfaces**, **StdBodies**, **Profiles**, **InterfaceRefs**, and **SkillRefs** are placeholders for multiple definitions of single elements.

6.2.2.2 Modelling of Variable Details in the Emplacement / $\text{Profile}_{\text{Empl}}$

Figure 6.3 illustrates how the Emplacement models and implements the Variable Details from the Generic Model. The BP follows a similar implementation. The lines 1 to 8 belong to the generic part of the Emplacement. Thus, these are shared and can be referenced by all $\text{Profile}_{\text{Empl}}$s belonging to that Emplacement. The lines 9 to 23 belong inside an imaginary $\text{Profile}_{\text{Empl}}$ characterising its use. These characteristics are focused on in detail below.

The Selective Properties (Definition 8) are defined in lines 9 to 11. The **VariableRef** on line 10 defines a mandatory Selective Property with id = *propSel.amount.fingers*. It specifies that all implementations of this $\text{Profile}_{\text{Empl}}$ will have a selective property with name *Amount of Fingers* and they will all have exactly two fingers. This information is composed from the description as follows. Line 10 defines, with reqOpt, that the Selective Property with a given id/idRef is mandatory. It must always be present. The other option for this attribute would be "OPT", stating that its parent is optional and can be missing. The idRef links this Selective Property to line 3, where jointly shared information about this property is defined. The human-readable name and description is given there, as are the given attributes refining and restricting the value, namely the datatype giving the formatting for the value data, the decade, the increment and the unit as given in Ch. 5.3.1. The attribute unit here demonstrates a case where the unit is un-defined. This is marked with the value "NA" meaning "Not Available". Otherwise the SI unit symbols [60] are used as values for the unit. In the case of datatypes, the implementation utilises the data type definition of IEC 61131-3 [52]. Finally, line 10 defines the value of the property stating the amount of fingers. As this is a Selective Property, its value will be constant across all implementations of this $\text{Profile}_{\text{Empl}}$.

Next, Properties (Definition 7) are specified in lines 12 to 23. The implementation classifies three different Property categories - business, environmental, and technical. The first collects together business, performance, reliability, and quality-related properties. The second collects environmental conditions, such as temperature, humidity and vibration-related properties in different operating states. The third captures the technical characteristics of the module. In the case of Properties, the $\text{Profile}_{\text{Empl}}$ defines all features of the property except the value. This is because the value will be defined later by the implementation of the MD.

The **BusinessProperties** element in lines 13 to 16 illustrates the definition in the case of two or more parallel properties under a collective element. Furthermore, the definition illustrates defining a property on the spot, rather than using an id reference pointing to definitions made in a shared section, even though this would be possible. The details of **BusinessProperty** are

```
1  <SharedResources>
2    <SelectiveProperties>
3      <VariableNum datatype="UINT" id="propSel.amount.fingers" name="Amount of
       Fingers" description="Amount of Fingers in the module" decade="0" increment
       ="1" unit="NA"/>
4    </SelectiveProperties>
5    <Properties>
6      <VariableNum id="prop.stroke" name="stroke" datatype="LREAL" description="
       Stroke of the gripper" reqOpt="OPT" min="0" increment="0.00001" unit="m"/>
7      <VariableNum id="prop.time.closing" name="closingTime" datatype="LREAL"
       description="Closing time of the gripper" reqOpt="OPT" min="0" increment="
       0.0001" unit="s"/>
8    </Properties>
```

```
9      <SelectiveProperties>
10       <VariableRef reqOpt="REQ" idRef="propSel.amount.fingers" value="2"/>
11     </SelectiveProperties>
12     <Properties>
13       <BusinessProperties>
14         <BusinessProperty classOfProperty="PERFORMANCE" id="propBus.operation.
       MDT" name="Mean Down Time (MDT)" description="The average time to repair a
       machine after a failure, it's also called MTTR (mean time to repair)."
       datatype="TIME" reqOpt="REQ" min="0" unit="h"/>
15         <BusinessProperty classOfProperty="PERFORMANCE" id="propBus.operation.
       MTBF" name="Mean-time Between Failure (MTBF)" description="The mean time
       between two consecutive failures of a system." datatype="TIME" reqOpt="REQ"
        min="0" unit="h"/>
16       </BusinessProperties>
17       <EnvironmentalProperties>
18         <EnvironmentalProperty classOfProperty="NORMAL_OP" id="propEnv.op.temp"
        name="Temperature during operation" description="Acceptable temperature
       range during the Operation" datatype="LREAL" reqOpt="REQ" unit="deg"
       increment="0.1"/>
19       </EnvironmentalProperties>
20       <TechnicalProperties>
21         <VariableRef idRef="prop.stroke" reqOpt="REQ"/>
22         <VariableRef idRef="prop.time.closing" reqOpt="REQ"/>
23       </TechnicalProperties>
```

Figure 6.3: Simplified example - excerpt of XML implementation of variables

revealed by focusing on line 15. Here, the property is identified by an `id` and its human-readable information is captured by a `name` and a `description`. Attribute `reqOpt` states that this property must always be present, the `datatype` format is time, the value is restricted with a minimum value (`min`), and the `unit` for this property is hour, marked with the symbol "h". Additionally there is a further classifying attribute, `classOfProperty`, which provides an additional grouping for making collections of similar kinds of properties. The **`EnvironmentalProperty`** on line 18 behaves the same way and follows the same structure as **`BusinessProperty`**. The illustration additionally restricts the resolution of the value with an `increment`.

The **`TechnicalProperties`** can follow the same description pattern for defining a **`TechnicalProperty`** as those used in **`BusinessProperty`** or **`EnvironmentalProperty`**, but referencing is used in this illustration instead. There could also be a mixture of both direct and referenced definitions. Referencing follows the same pattern as shown in the case of Selective Property described above. **`VariableRef`**, on line 21, defines the existence of a Property and `idRef` links it to the shared definition on line 6, which characterises the property. Here, the double definition of `reqOpt` attribute should be noted. The common part defines it as optional

(line 6). This is the default cardinality that would be used if the actual place of use (line 21) makes no definition of the cardinality at all. However, the definitions made at the actual place of use have higher priority than the common definitions. Therefore, in this case the property must always be present, as defined in line 21.

6.2.2.3 Modelling of Capabilities and Skills in the Emplacement/Profile$_{\text{Empl}}$

Figure 6.4 illustrates how the Emplacement models and implements the Capabilities and Skills from the Generic Model (See Ch. 5.3.2). The starting point of this example will be the **`Profile`** on line 10. In lines 12 to 15, references are made to two skills which are implemented by this profile. Focusing on line 13, **`SkillRef`** references the skill *Open* with `idRef`. It further defines that this skill is mandatory for this **`Profile`**, with the `reqOpt` value.

The actual definitions of the skills are collected under **`ControlIFs`** (lines 18 to 38). Following the previous skill pointer brings us to line 19, where the actual **`Skill`** for open action is defined. Here the skill is given its `id`, `name`, `description`, and the `class` which is used to categorise the class of skills. Element **`ParentSkills/SkillRef`** can be used to create skill hierarchies. The lower level skill references its parents. **`Mappings/CapabilityMapping`** can be used for mapping the **`Skill`**s to capabilities, stating that this skill is implementing another capability. In a similar way, the elements **`Mappings/SkillMapping`** can be used to make similar connections to other skills. The **`ControlPorts`** (lines 22 to 29) is the implementation of the event flow of the Generic Model. It defines the input ports (**`Inputs`**) and output ports (**`Outputs`**) for events of this skill, and associates variables to each type of port. The variable can be defined on the spot or referenced to common variable definitions made in the shared part of the Emplacement (lines 1 to 8 in the example). The event flow (**`ControlPorts`**) is followed by the **`ParameterPorts`**, which implements the data flow of the Generic Model. Here, a similar internal structure is followed as in **`ControlPorts`**.

The variables can be defined in two different places - on the spot where they are used or in the **`SharedResources`** section. In the latter case, the variable is available throughout the entire Emplacement, and can be referenced with **`VariableRef`** element with the corresponding `idRef` value (e.g. line 33 referencing to line 3). **`VariableRef`** has `reqOpt`, defining whether the variable is mandatory or optional. The definition of a variable has four different options in the implementation. These are **`Variable`**, **`VariableBoolean`**, **`VariableNum`**, and **`VariableString`**. **`Variable`** defines the basic set of attributes, which are then extended with a few additional data type-specific attributes in the other three cases. The basic set of attributes consists of the `id`, `name`, `description`, `reqOpt`, `datatype`, `increment`, `decade`, and `unit`. The other three add attributes such as `value` and `defaultValue`, and in the case of a numeric variable, minimum value (`min`) and maximum value (`max`) are also added.

6.2.3 Blueprint

This section introduces an implementation of the Module Description (See Ch. 5.2.2). The Blueprint file format is specified in [25].

Definition 14: Blueprint

The Blueprint (BP) is an implementation of the Module Description (MD) (See Definition 5). The BP (file) is a digital specification of a physical production module. Its format is XML, according to the given XSD. It is supplied by the module vendor and is delivered together with the module. The BP file content needs to adhere to the Profile$_{\text{Empl}}$ and

```
<SharedResources>
  <Variables>
    <VariableNum id="var.time.opening" name="OpeningTime" datatype="LREAL"
    description="Opening time" reqOpt="OPT" min="0" increment="0.01" unit="s"/>
    <VariableNum id="var.time.closing" name="ClosingTime" datatype="LREAL"
    description="Closing time" reqOpt="OPT" min="0" increment="0.01" unit="s"/>
    <VariableBoolean id="var.action.trigger" name="trigger for action" datatype
    ="BOOL" reqOpt="REQ" unit="NA" description="Trigger for this action"/>
    <VariableBoolean id="var.action.done" name="Action is done" datatype="BOOL"
     reqOpt="REQ" unit="NA" description="Action is completed"/>
  </Variables>
</SharedResources>
<Profiles>
  <Profile moduleLevel="UNIT" id="prof.gripper.example.1" name="Example Profile
     with attaching gripper according ISO 29262" version="1.0" description="
    This is the profile defining the generic attaching interfaces and
    properties for grippers." isAbstract="true">
    ...
    <SkillRefs>
      <SkillRef idRef="act.open.1" reqOpt="REQ"/>
      <SkillRef idRef="act.close.1" reqOpt="REQ"/>
    </SkillRefs>
  </Profile>
</Profiles>
<ControlIFs>
  <Skill id="act.open.1" name="Open" description="This action will open e.g.
    the jaws of gripper." class="ACTION">
    <ParentSkills/>
    <Mappings/>
    <ControlPorts>
      <Inputs>
        <VariableBoolean id="var.doOpen" name="doOpen" datatype="BOOL"
    description="Triggers the opening action" reqOpt="REQ" unit="NA"/>
      </Inputs>
      <Outputs>
        <VariableBoolean id="var.openDone" name="openDone" datatype="BOOL"
    description="Indicates that opening has been performed" reqOpt="REQ" unit="
    NA"/>
      </Outputs>
    </ControlPorts>
    <ParameterPorts>
      <Inputs/>
      <Outputs>
        <VariableRef reqOpt="OPT" idRef="var.time.opening"/>
      </Outputs>
    </ParameterPorts>
  </Skill>
  <Skill id="act.close.1" name="Close" description="This action will close the
    jaws of gripper." class="ACTION">
    ...
```

Figure 6.4: Simplified example - excerpt of XML implementation of capabilities and skills

Emplacement specification of which the BP is an instance. No conflicts are allowed. The BP must contain pointers to its parents – the Emplacement and $\text{Profile}_{\text{Empl}}$. This link can be used for validating the content of the BP against its parents.

A module vendor has to provide a BP file for every production module available for a modular production system. This file gives the description of the implementation. It is always a representation of one $\text{Profile}_{\text{Empl}}$ of one Emplacement, and it shows which classes and options are selected from the $\text{Profile}_{\text{Empl}}$ and binds those with the selected values. The BP file gives the values for the module-specific Properties identified by the Emplacement. The BP file acts as the input for a system integrator to design an assembly system based on selected criteria. Modules which belong to the same Emplacement and the same $\text{Profile}_{\text{Empl}}$ inside the Emplacement are interchangeable as long as they both implement the same interfaces, classes and options. Such production modules allow a direct comparison of costs and performances, even if they originate from different vendors.

Both the Emplacement and BP XML files (as well as the XSDs specifying the data models) are in machine-readable form and express all the available features of the production module. The Emplacement and BP files are utilised by the software tools and search engines for assembly system layout design, for assembly system simulation, for system configuration, and for control purposes.

The information about both the Emplacements and the BPs is only stored in one place in an agreed format. This (XML) ensures maintenance, coherency, error avoidance, and comprehensibility for the SW tools. When human-readable documentation, or any other kind of documentation or view is needed, it should be generated from this single source of information.

6.2.3.1 Modelling the Interfaces in Blueprints

The interfaces are one of the most important aspects defined by the BP document. The interfaces determine which modules can be connected together, and how and where. The implementation contains a few different coordinate frame types. Each interface, or a part of it, in which can be mounted a freely selectable counterpart, is represented with a unique Interface Frame (F_i). The Body Frame (B_i) is fixed on each moving body of the production module, relating to the solid kinematics and representing this body's origin.

A natural and commonly identifiable position in the module is selected as the *local origin* (O_L). This frame is a special one and it defines the pose of the module's origin. All measures of the module and all other module-related frames are given relative to this frame. Thus, this pose is also the main Body Frame, B_0, for the module. In the context of this thesis, B without an index is taken to be the equivalent to B_0. Although it is preferable that one of the interface frames (F_i) within a production module (i.e. the BP) is selected as the O_L, this is not mandatory. Global Frame (G) is the production cell's global origin. The natural choice for G would be the O_L of the base framework module of the cell.

For relations between the frames hold:
The local origin of a production module satisfies.

$$^{G}\mathbf{r}_{O_L} \equiv O_L = B_0 \equiv B \tag{6.1}$$

In scope of a module holds that all interface frames for a module are defined relative to module's local origin.

$$\forall F_i : [\, \exists^{B}_{O_L} \mathbf{r}_{F_i} \;\equiv\; \exists\overrightarrow{O_L F_i}\,] \tag{6.2}$$

The same hold for the body frames.

$$\forall B_i : [\, \exists^{B}_{O_L} \mathbf{r}_{B_i} \;\equiv\; \exists\overrightarrow{O_L B_i}\,] \tag{6.3}$$

```
1  <Interface id="MECHANICAL_PORT-IF1-std.TUTuF.0002-F">
2    <InterfaceRef interfaceRef="std.TUTuF.0002" description="Process module
   interface" interfacePortRef="IF1" ifType="MECHANICAL_PORT" gender="FEMALE"
   reqOpt="REQ" minOcc="1" maxOcc="1">
3      <Class name="Module Height" values="2" ifType="MECHANICAL_PORT"
   description="Height multiplier of the module for stacking. N= no upper
   interface, rest n x 50mm."/>
4    </InterfaceRef>
5    <ForceAndTorqueLimits increment="0.1" decade="0" unit="N" rot_increment="
   0.1" rot_decade="0" rot_unit="Nm">
6      <Min x="-0" y="-0" z="-0" rot_x="-0" rot_y="-0" rot_z="-0"/>
7      <Max x="0" y="0" z="0" rot_x="0" rot_y="0" rot_z="0"/>
8    </ForceAndTorqueLimits>
9    <Instances>
10     <Instance reqOpt="REQ" id="MECHANICAL_PORT-IF1-std.TUTuF.0002-F-1"
   interfacePortRefQualifier="1">
11       <Location x="0.0" y="0.0" z="0.0"/>
12       <Orientation angle="0.0" x="0.0" y="0.0" z="0.0"/>
13     </Instance>
14   </Instances>
15 </Interface>
```

Figure 6.5: Simplified example - excerpt of XML implementation of a static mechanical interface

Figure 6.5 shows a short extract from the BP implementation for a static interface definition. It represents the mounting interface of a TUT μFactory manipulator module, which is at the same time the local origin (O_L) of this same module. This can be seen from lines 11 and 12, as all the values are zeros. The interface is illustrated as B_0 in Figure 6.6.

What else is modelled with this interface definition? Line 1 in Figure 6.5 is the start of an **`Interface`** definition and it defines its unique identifier (`id`). The interface can have 1 to n implementations in different interface frame poses within the module. These are modelled with **`Instance`** -elements (lines 10 to 13), and each implementation is also given a unique identifier (`id`). The identifiers can be used to reference a specific interface or an implementation of the interface from other parts of the BP (e.g. in case of naming the local origin (O_L) for the module) or by tools using module descriptions. The interface port instance in line 10 is denoted in this example as F_1.

The **`Location`** and **`Orientation`** are used to specify the physical pose of the interface implementation. The former denotes the position vector (${}^B\mathbf{r}_{F_1}$) of the implemented interface frame (F_1) within the module's local coordinate system in meters. The latter defines the rotation of the F_1 relative to the O_L. The concept of axis-angle [5] is used for the description, which defines the frame rotation with four real numbers. The `angle` is given in radians and the other three values define a unit vector (${}^B\hat{u}$) around which the rotation takes place.

There are four different means for representing the 3D rotation, i.e. the frame orientations. These are axis-angle, rotation matrix, euler angles, and quaternions [5]. The reasoning behind selecting this method for the BP are as follows. The objective is to have an exact representation of the orientation with the minimal amount of numbers. The rotation matrix is simple to understand, but it uses nine numbers for the representation. The Euler angles are discarded because of exactness, as the order of applied rotation angles is significant and creates a high potential risk for confusion. The quaternions also use four numbers, but the physical meaning of the representation remains open. Thus, the axis-angle was selected as the best representation. However, the selection does not block out any of the other representations, because there are direct transformations from one representation to another which can be utilised in the implemented application.

The **InterfaceRef** (lines 2 to 4) creates links to other parts of the BP document and defines the interface further. It follows the definitions made by the parent $\text{Profile}_{\text{Empl}}$, and cannot conflict with it in any way. It links to the specification of the followed interface standard by having the id of the standard as the value of `interfaceRef`. In the same manner, it links the interface to an abstract interface port (`interfacePortRef`). The purpose of this is to harmonise different implementations of the same Emplacement, and to provide a semantic link between these implementations. For example, one abstract interface port could be the mounting interface for a module. Different interface standards can be applied (by other BPs), but they are all still a mounting interface in the abstraction sense. This can be realised from all implementations pointing to the same abstract interface port i.e. `interfacePortRef` = IF1. The `IfType` defines the classification for the purpose of the interface. The values for classification are mechanical, electrical, service, and communication. The gender of the interface is defined with `gender`, having the values male, female, and neutral available. `MinOcc` and `maxOcc` specifies the accepted amount of instances in the implementation. Finally, the `description` gives human-readable documentation for the interface. The attribute `reqOpt` functions according to the globally given manner.

The **InterfaceRef** may have multiple sub-elements of **Class** and **Option**. These usually formalise textually defined features from the interface standard specification. In the case of the Emplacement, the `values` attribute defines a list of all the accepted values. In contrast, in the case of the BP, the `values` selects the class or option value, (or occasionally the values) which best fits with the implementation of the physical module, and removes the rest.

The **ForceAndTorqueLimits** (lines 5 to 8) defines both the minimum and maximum values for the forces and torques that can be applied to this interface. These values are represented in the interface frame (F_1). The units, increments, and decades are defined under the main element (**ForceAndTorqueLimits**) for both the linear and rotational axes. These are applied to all associated linear or rotational values under the child elements, according to the pattern $value \times 10^{decade}\ unit$ (Eq. 5.11).

6.2.3.2 Modelling the Kinematics in BPs

The interfaces are not only stationary, but can also represent dynamic interfaces, such as the tip of a manipulator. The AMD or Emplacement does not consider the kinematics model of the production module, but represents bodies with stationary interfaces. In the case of the BP, the kinematics model of an interface is added to the data model of the production module. A Denavit-Hartenberg (DH) parameter set is used to represent the kinematics model (See Ch. 3.4.5.2 (p.51)).

DH parameters have been selected as the data model for kinematics because: a) it is a well recognised method for expressing kinematic structures in both the literature and in the robotics industry, b) the generalisability of the solution, i.e. it can express different kinds of kinematic structures both in amount and type. In the former it can relate to an unlimited number of links, and in the latter it has different kinds of prismatic and revolutionary joints, which are connecting links in arbitrary configurations; and c) the compactness of representation.

The TUT μFactory gantry type manipulator module, with 3 DOF driven by linear motors is used to demonstrate the use and modelling with DH parameters. The manipulator axes are in a Cartesian configuration Y-X-Z. The DH parameter set and frame positions are determined. These are illustrated in Table 6.2 and Figure 6.6.

Lines 12 to 28 in Figure 6.7 show how the DH parameter set for this specific manipulator module is modelled in practice, by providing an extract from the module's BP file. The data model of a kinematic interface follows the same structure and definitions as those used for a static interface, but adds two new child elements to it. Compared to Figure 6.5, Elements **DH_set** and

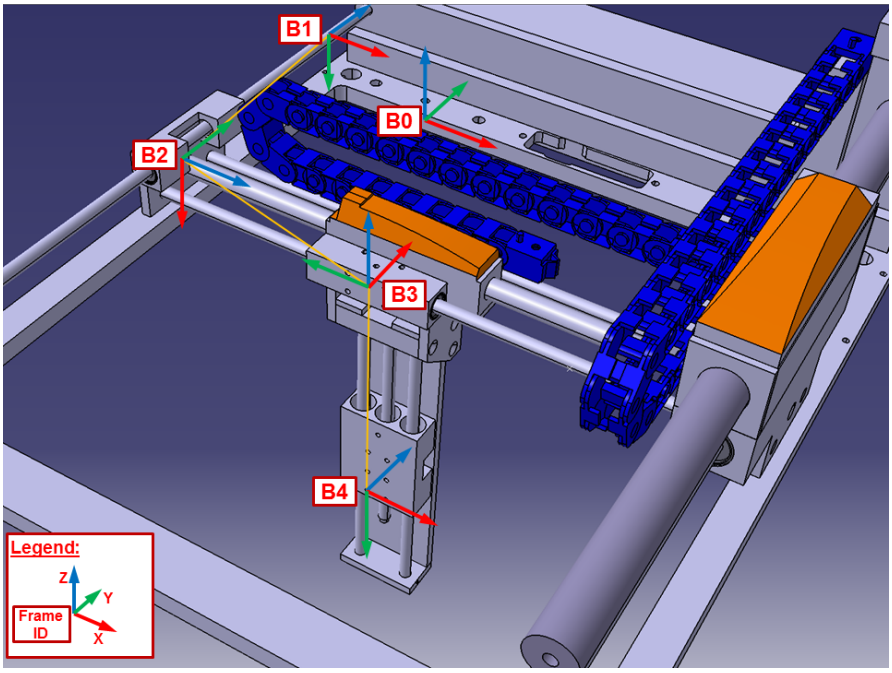


Figure 6.6: TUT μFactory Linear manipulator - Kinematic Frames

Table 6.2: DH parameter set for TUT μFactory linear motor manipulator

Frame No.	a_i [mm]	α_i [deg]	d_i [mm]	θ_i [deg]
1	-30,751	-90	25,027	0
2	9,75	90	-165 + P_Y	90
3	-7,5	-90	15 + P_X	90
4	0,0	-90	-131,5 + P_Z	-90

, where
P_X = Position of X-axis. Value range: 0..145 mm.
P_Y = Position of Y-axis. Value range: 0..135 mm.
P_Z = Position of Z-axis. Value range: 0..72 mm.
a_i = link length
α_i = link twist
d_i = joint distance or link offset
θ_i = joint angle

`Parameters` are added to the **`Interface/Instances/Instance`**. The interface instance on line 8 is denoted as F_{21}.

The **`DH_set`**, lines 12 to 23, contains $\mathbb{N}\times$ **`DH_param`**s, each representing a row in the DH model [66]. The **`DH_param`** contains `rowNumber` and all four DH parameters: `linkLength` (a_i), `linkTwist` (α_i), `linkOffset` (d_i), and `jointAngle` (θ_i). The parameters are given in units of meters and radians, according to the type of parameter. The `rowNumber` defines the index i and the sorting order for the parameters. Each **`DH_param`** may contain dynamic components, which are the position or angle of the real axis moving the kinematic structure. These are represented with **`DH_param/Parameter`**. It contains: a) `associatedWith` linking this parameter to one of the four DH parameters; b) `parOperator` defining whether the value of the given variable is

```
<Interface id="MECHANICAL_PORT-IF2-std.Proprietary-F">
  <InterfaceRef interfaceRef="std.Proprietary" reqOpt="OPT"
interfacePortRef="IF2" ifType="MECHANICAL_PORT" description="Proprietary
attachment IF for endeffectors" gender="FEMALE" minOcc="1" maxOcc="1"/>
  <ForceAndTorqueLimits increment="0.1" decade="0" unit="N" rot_increment="
0.1" rot_decade="0" rot_unit="Nm">
    <Min x="-0" y="-0" z="-0" rot_x="-0" rot_y="-0" rot_z="-0"/>
    <Max x="0" y="0" z="0" rot_x="0" rot_y="0" rot_z="0"/>
  </ForceAndTorqueLimits>
  <Instances>
    <Instance reqOpt="REQ" id="MECHANICAL_PORT-IF2-std.Proprietary-F-1"
interfacePortRefQualifier="1">
      <Location x="-0.091" y="-0.182" z="-0.116"/>
      <!--Lowest hole, Z-pointing to hole, X-to adjasent hole. X keeps its
orientation. Negative Y pointing up-->
      <Orientation angle="1.570" x="-1.0" y="0.0" z="0.0"/>
      <DH_set>
        <DH_param linkOffset="0.025027" linkTwist="-1.570796" linkLength="
-0.030751" jointAngle="0" rowNumber="1"/>
        <DH_param linkOffset="-0.165" linkTwist="1.570796" linkLength="
0.00975" jointAngle="1.570796" rowNumber="2">
          <Parameter idRef="dhPar.position.Y" parOperator="PLUS"
associatedWith="LINK_OFFSET"/>
        </DH_param>
        <DH_param linkOffset="0.015" linkTwist="-1.570796" linkLength="
-0.0075" jointAngle="1.570796" rowNumber="3">
          <Parameter idRef="dhPar.position.X" parOperator="PLUS"
associatedWith="LINK_OFFSET"/>
        </DH_param>
        <DH_param linkOffset="-0.1315" linkTwist="-1.570796" linkLength="0"
 jointAngle="-1.570796" rowNumber="4">
          <Parameter idRef="dhPar.position.Z" parOperator="PLUS"
associatedWith="LINK_OFFSET"/>
        </DH_param>
      </DH_set>
      <Parameters>
        <Parameter id="dhPar.position.X" name="Gantry_position.X"
description="Position of gantry in X direction" datatype="LREAL" min="0"
max="0.145" defaultValue="0.010" unit="m"/>
        <Parameter id="dhPar.position.Y" name="Gantry_position.Y"
description="Position of gantry in Y direction" datatype="LREAL" min="0"
max="0.135" defaultValue="0.130" unit="m"/>
        <Parameter id="dhPar.position.Z" name="Gantry_position.Z"
description="Position of gantry in Z direction" datatype="LREAL" min="0"
max="0.072" defaultValue="0.070" unit="m"/>
      </Parameters>
    </Instance>
  </Instances>
</Interface>
```

Figure 6.7: Simplified example - excerpt of XML implementation of a dynamic mechanical interface and its DH parameters

added ('PLUS') or subtracted ('MINUS') from the constant part represented by the model, and c) `idRef` linking it to the variable definition.

The **`Parameters`**, lines 24 to 28, contain $\mathbb{N}\times$ **`Parameters/Parameter`**s, each defining a variable in the controls, representing the actual position or angle of an axis. The attributes for a **`Parameter`** follow the same definitions as Variable Details do for the Profile. Noticeable are `min` and `max`, which define the range through which the axis is able to move, and `defaultValue` which defines the default position for the axis. The final value of a dynamic DH parameter is the result of an arithmetic operation (**`DH_param/Parameter`**@`associatedWith`) applied to the constant part derived from **`DH_param`**, and the current (or simulated) value of the axis variable assigned by **`Parameters/Parameter`**.

The pose of the interface in the static case, i.e. **`Location`** and **`Orientation`**, are defined in the default position of the kinematic structure. This is a natural position for the kinematic structure, such as home position in this case. The module vendor may freely choose the default position. The same position is achieved applying the `defaultValue`s from the **`Parameters`** of the kinematic model.

6.2.3.3 Modelling the Capabilities and Controls in BPs

The data model of capabilities and skills for the BP is, as far as possible, the same as in Emplacement and $\text{Profile}_{\text{Empl}}$, as presented in Ch. 6.2.2.3. The BP extends the former data model with a controller implementation-related section. The **`Implementation`** contains additional information about the controller interfacing and symbols, and their access locations. The concepts are represented and used in the same way in both cases, and only the number of skills and capabilities may differ. The implementation of the production module may restrict the feature set, if the $\text{Profile}_{\text{Empl}}$ defines a skill as optional. All mandatory capabilities and skills must be present in the BP in every case, in order to achieve compliance.

The feature set of $\text{Profile}_{\text{Empl}}$ can be extended, if the production module contains additional functions. However, in this case, there must not be any conflict with any of the mandatory or optional capabilities or skill definitions made in $\text{Profile}_{\text{Empl}}$. The visibility of these added capabilities is limited. When the system design is done at the AMD level, the information from Emplacement is used. The additions made at a lower level will not be acknowledged at this design phase, but only later in the detailed design. Thus, it is important to propagate all common capabilities and skills to $\text{Profile}_{\text{Empl}}$.

The set of capabilities and skills can be formalised for $\text{Profile}_{\text{Empl}}$s:

$$C_P = C_P^m \cup C_P^o \tag{6.4}$$

And capabilities for Blueprints (C_{BP})

$$C_P^m \leq C_{BP} \leq (C_P^m \cup C_P^o \cup C_{BP}^a) \tag{6.5}$$

, where

C = the set of capabilities and skills

C^m = mandatory C's

C^o = optional C's

C^a = additional C's

Sub-indexes:

P = $\text{Profile}_{\text{Empl}}$

BP = Blueprint

6.2.4 History Container

This section introduces the implementation of module instance information (See Ch. 5.2.3).

Definition 15: History Container

The History Container is the implementation of the Module Instance Description (Definition 6). It presents information associated with the specific instance of the production module, which cannot be generalised to the BP. It appends the module information defined in the BP, and can override the BP's property values.

The History Container (HC) file format is designed in the first version, but not documented as standard. A rough outline of the content is presented here. First, it contains the MID identification information and description. This is followed by the pointers to its parents, BP, $\text{Profile}_{\text{Empl}}$, and Emplacement. Next come the placeholders for the four main content carries: **`Owners`**, **`Connections`**, **`Events`**, and **`PropertyUpdates`**. Finally, there are elements collecting together the **`Actors`** and **`SharedResources`**, i.e. the common Variable Details information.

The main container has a number of different elements. These include the Element **`Owners`**, which contains a listing of all the owners of the module and the locations where the module has been. **`Connections`** contains the past connections of this module. It lists which interface was used and to which module the connection was made. It also stores the `status` of the connection, i.e. success or failure. This is important cumulative knowledge which can later be compiled and used when designing future systems. The time period and the actor performing the operation are stored. The **`Events`** element stores general events and activities related to the module. Examples of these are: attach or detach to system, usage hours, storing the module, maintenance and calibration actions, or general failure. An event also stores the description, the duration of the event, and the actor preforming the operation. **`PropertyUpdates`** stores updated property information, specific for this instance. These values override the values given in the BP, which are common to all modules of same type. Property updates are needed in cases associated with wear and tear, statistical data, and changes in the business environment. Each record is time-stamped and associated with the performing actor.

6.3 General Features of the Developed Description Formats

This section presents general features and concepts associated with the implementation of the Emplacement Concept and associated description formats.

Verification and Validation of Descriptions

The document verification is performed mainly through checks offered by XML and XSD. The XSD is used to define and verify the syntax and semantics of the exchanged description documents. It can be used by reference to automatic tools and SW applications, which then perform the verification process or other desired actions, such as generating an initial file as the starting point for a new definition. There are five distinct verification phases within the Emplacement Concept and implementation.

The first phase of verification is to check that the production module description (document) is in the condition the provider intended it to be. This is to ensure the data integrity and to capture any possible data corruption or tampering with the description. The use of check sums is intended to correspond to this requirement (UR 11.a), in this implementation. A check sum is calculated with a hash function for each published description, and this check sum is then published together with the description. The receiver may then recalculate the hash sum for the received description, and compare it with the one calculated by the provider. If the hash sums match, the description is intact and can be safely used. Otherwise the description may not be trusted. The reasons for this could be corruption during storage or exchange, different versions of the description, or even, in the worst case, that it has been tampered with. SHA-2 (specifically SHA-256) [94] has been selected as the method for calculating the hash sums. This is because: a) the method is well known; b) there are tools available for calculating it; c) it has not yet been broken, and d) the hash code is a reasonable size. The condition c) rules out older methods such as MD5 and SHA-1. Condition d) places the preference on SHA-256 rather than on SHA-384, SHA-512 or other larger ones. Digital signatures using public key infrastructure would provide better protection as a means of authenticating the information provider. However, it would create issues because of the additional complexity, the necessity for certificates, providing platform independence for the solution, and the effort involved in managing the private keys. For the time being, hash sums are considered to provide an adequate level of protection. This is because both the descriptions and the check sums are distributed from the same publicly available server, meaning that: a) the content is under the control of the administration, and b) the data can be retrieved and re-checked at any time.

The second phase of verification is to check that a description file is *Well-formed*. In this phase the XML document is checked against the W3C standard for XML, which verification can be provided by almost any generic XML tool. This ensures, for example, that: a) the content of the document is appropriately defined; b) It is delimited by the start and end tags, which define an element and its content; c) the tags are nested properly i.e. there is no missing end tag nor any overlapping tags; d) case sensitivity is respected within tags; e) attributes are represented with the correct syntax inside an element; f) the Characters are legally encoded, and g) the XML document has only a single root element.

The third phase of verification is the semantics check using XSD. The main advantage of using XSD is that both the sender and receiver of the information know that the files meet a certain quality level. In addition, the tools are able to read it in an interpretable form as the structure, content of the typed fields (dateTime, integer, string), and the fields with enumerated values will all be set according to the rules, and mandatory parts are always present. All this assumes that the sender has performed the XSD verification check before submission, and it has passed. All in all, the use of XML and XSD will provide great benefits for the integrity of the concept as the methods and tools are standardised (by W3C), and the verification process can be automated and easily integrated into user applications. In addition, it will detect quite a number of errors which might not otherwise be found, at least not easily. This type of verification process is illustrated in Figure 6.8 by verification processes which are associated with checking whether an Emplacement or BP document is valid.

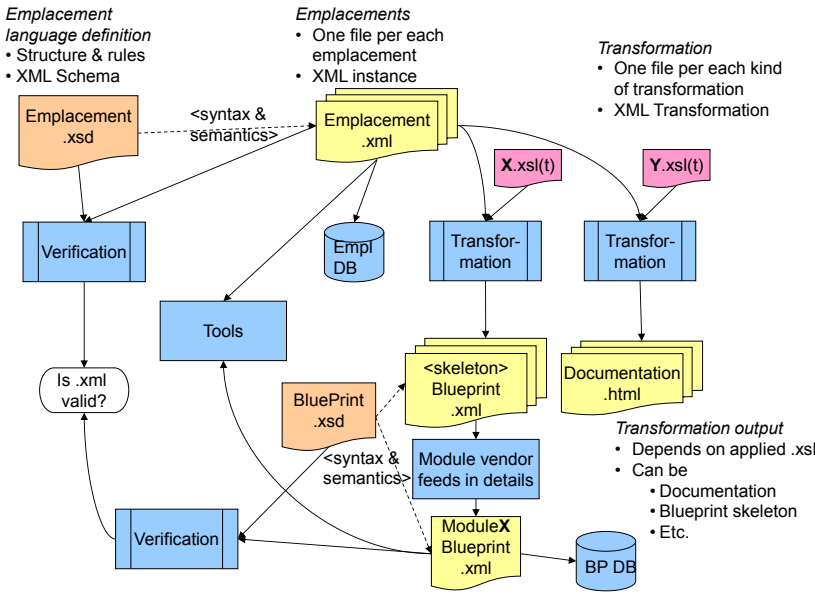


Figure 6.8: Relations of implemented file formats and associated processes

The XSD file is the main and the primary source for defining the model and content of the description documents. A second source is the manual of the XSD file (automatically generated document), and a third is the interface standard document. This order of precedence is used to resolve possible conflicts in the description formats. This is because all the rules, syntax, and semantics are described in greatest detail in the XSD files, while the rest are manually maintained according to the primary source. All version updates or corrections to errors are first applied directly to the XSDs.

However, these two last verification phases (well-formed and validity) will not ensure that the content of the files is right and valid throughout the Emplacement Concept. This can only be validated with another kind of review process. Thus, a fourth phase of verification is needed, verification between the peers, or against the description at a higher level in the hierarchy.

Three different cases can be determined for the fourth phase - one for each description file level. In the case of the Emplacement documents, the extra verification steps can be between the peers. The verification focuses, for example, on the consistent use of referencing, standards, interfaces, and capabilities; and the avoidance of duplicate definitions for the same things. In the case of the BP, the verification in this fourth phase is made against the Emplacement document and $\text{Profile}_{\text{Empl}}$, of which this module is an instance. The BP can be verified by checking that: all required matters do exist and are compliant with the superior document; the IDs used in Emplacement and $\text{Profile}_{\text{Empl}}$ are found in the same form in the BP document; the content matches with $\text{Profile}_{\text{Empl}}$; the given cardinalities are respected; the standards are presented as in the superior documents, and the optional structures are as specified. In the same way, the HC document can be verified against its superiors, its BP and its Emplacement/$\text{Profile}_{\text{Empl}}$.

In all cases of the fourth phase, reading in multiple files is required and comparing them according to separately defined rules. This all requires a dedicated application for performing the verification. However, the good thing is that the verification can be automated, and that the application(s) and its rules apply at the data model level. Therefore, different descriptions can be read in, processed, verified and judged without a change in the defined rules. The Schematron

[59, 175] may play a role in providing a technical solution for the potential problems detected in the fourth phase verification. It would provide a method for describing the rules and assertions for validating structures between multiple files or patterns inside a file, and would offer processing capabilities specifically for this case.

The fifth phase of validation is the ultimate verification and validation, which requires yet another kind of review process. Unfortunately, it is not as straightforward as the four earlier phases. This is a process for the module provider to ensure and validate that the module performs and operates as described by the MD. There could be a certification or self-certification procedure for this phase. This verification, like the earlier ones, must be repeatable at any time by, e.g. the end-user, to ensure that the module still meets its specification. Thus, the test procedures for validation must be well documented and repeatable. The MD may contain its own section for documenting the test and calibration procedures, as the BP file does. However, the tests might require specific test racks and jigs, and/or measurement applications, which might only be available from the module provider. Thus, this final phase of validation cannot yet be automated as easily as the earlier ones.

The current implementation provides solutions for the first three verification phases. Phases four and five have not yet been implemented, but are left for future work. These would require further research, which would ensue from the wider adoption of the Emplacement Concept.

7 Utilisation of the Formal Description Formats

This chapter has four main parts introducing utilisations of the proposed concept. The first section presents how the AMD/Emplacement and the MD/BP descriptions can be utilised from the system design and engineering perspectives. The focus is on the capabilities, the interfaces and the kinematics. The second section focuses on how the information in the BP descriptions can be converted into other formats, and what can be gained from this. One example is to provide more human-readable and a better formatted view of these descriptions by the use of the transformations. The third section presents a few tools associated with the concept, while the fourth part deals with Case Study 1. This is further divided into two sections - Ch. 7.4 presents the environment, the modules and their descriptions, and Ch. 7.5 presents how the Emplacements are created.

7.1 System Design and Engineering

This section discusses the significance of the descriptions for different stakeholders, and how they can utilise the proposed concept and the description in the system design and engineering processes. Particular attention is paid to the capabilities and the interfaces. This is not an exhaustive list of all possible use cases and utilisation scenarios, but it does highlight the most important and illustrative ones.

7.1.1 Impact of the System Design Process on Stakeholders (and associated Use Cases)

The main stakeholders in the concept are the module providers, the system integrators, the end users and the harmonisation organisation, as defined in Ch. 4.2.3. The users and their relationships are illustrated in Figure 4.1 (p.63) and in Figure 7.1. The following sections discuss in more detail the role of each stakeholder in the Emplacement Concept, the impact of the concept on these users, and how they can utilise the concept through the use cases. The scenarios can be generalised to the level of the Generic Model, thus the terminology of the Generic Model is used in this subsection.

7.1.1.1 Harmonisation Organisation (i.e. User Group)

The module providers, integrators and end users must have a mechanism to make additions and changes to the *Abstract Module Descriptions* and *Profiles*, because there will be new technologies, modules for new areas, and new processes appearing on the market. In order to make sure this specification process is performed in a controlled way, a central and neutral body is to maintain the AMD and Profile definitions. This organisation would also be responsible for harmonising and standardising the content of the AMD specifications. In Figure 7.1 that role is designated

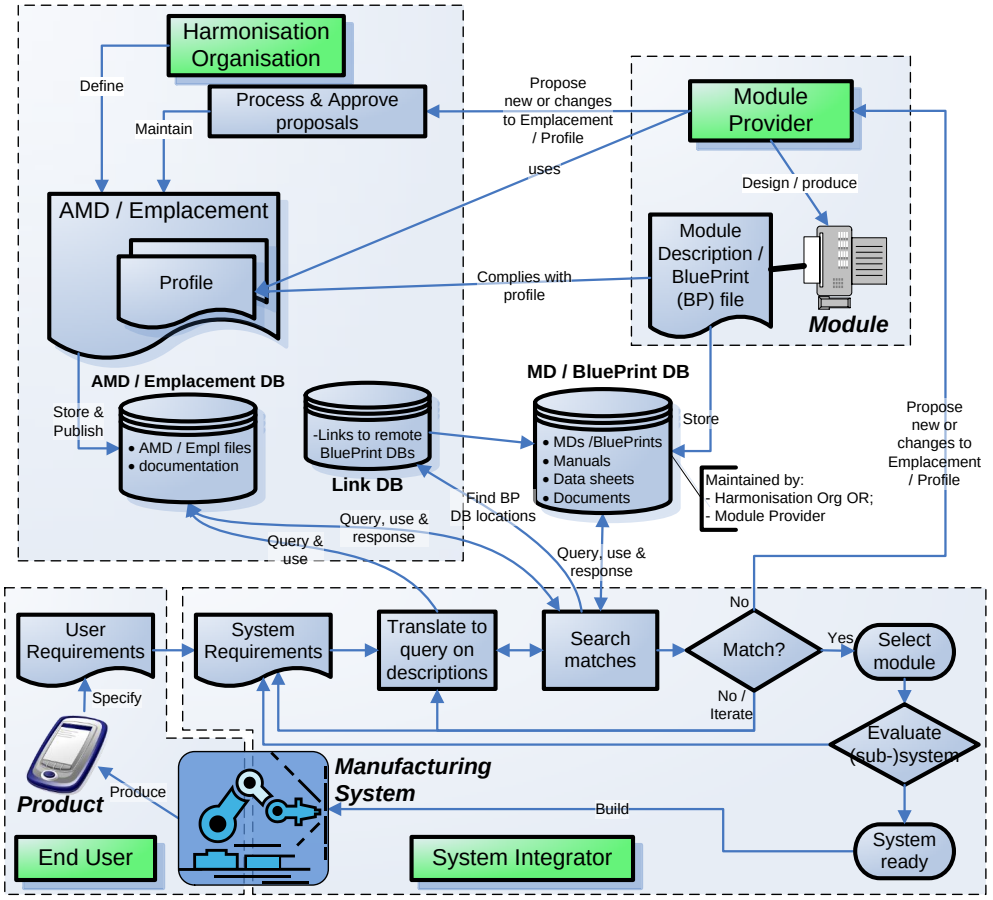


Figure 7.1: Emplacement Concept use-case and stakeholders. Modified from [26]

as Harmonisation organisation. In practise, the harmonisation organisation is the *User Group* promoting and advancing the Emplacement Concept and associated technologies. The end users and also the integrators should approach primarily the module vendor companies in order to trigger new initiatives or changes to the existing AMDs. The module vendors will act as a filter for the proposals.

The *User Group* is responsible for maintaining the AMD and the Link DB (See Figure 7.1). The user group can also host the MD DB(s), but those may be distributed to the module provider companies, or to an external broker. The Link DB will serve as a central entry point (like yellow pages) to find the available MDs that comply with the Emplacement Concept easily. The end users can utilise this linking service to access all MDs through one central location, without searching for the information themselves from the vendor home pages. One can compare this to the services and package system that the Linux distribution has for distributing software applications and packages.

7.1.1.2 Module Provider

The module providers make the implementations of their process modules according to the AMD and the Profile definitions. They need to fulfil the mandatory interfaces, properties and skills (i.e.

module capabilities) in their module design. Therefore, the AMD and the Profile selections need to be performed at the very earliest stage of the module design. Once the design is complete, they can generate and complete their MD, which will represent the features and characteristics with the defined values of the recently designed module. Some tools are available for MD file creation. The module provider can use Emplacement Web Service (EmplWS) [173] or MD editor to create a Skeleton MD file from the selected AMD and Profile, and start filling in module-specific information. The mentioned tools are presented later in Ch. 7.3. Once the implementation of the module is complete and there is an MD representing it, the module provider can publish the MD of their module for others to use.

Sometimes there is no suitable AMD and Profile available. In such cases, the module provider can propose a new Profile (and AMD) specification for the *User Group*. The module provider has two main alternative ways to create new Profiles: 1) new work item proposal for the user group with requirements, or 2) making first the specification of the module and create the MD for it, and then proposing it as the basis for a new Profile for the user group. [25] describes these different scenarios in detail, and possible side-effects.

The SMEs, out of all module providers, stand to gain the most from the Emplacement Concept, as it is aimed at providing self-contained and easily integrated modules. This allows the SME to focus on a specialised and innovative process, create a module around it, and still be able to enter markets easily, because the concept ensures that the module will be easily integrated into the overall system and can function as a part of it. The clear and precise module boundaries ease the testing and re-use of the modules, which can in turn lead to increased quality and sustainability of the modules. The concept makes it possible to use partner networks in the system creation. The SMEs, including both module providers and system integrators, can establish together a network where each company has its own role and process speciality. The shared concept enables fast and reliable system integration with clear responsibilities and delivery boundaries.

7.1.1.3 System Integrator and End User

The system integrators and the end users are the main users and potentially the greatest winners of the proposed concept. System designers and integrators can make better and faster decisions based on more thorough, correct, and valid information, which can be retrieved on-line. They would benefit immensely in terms of cost savings and quality improvements when they get the detailed descriptions of production resources in a common format, that is the MD/BP. This would eliminate the need to recreate and retype information (often again and again), and would also lead to savings in time and a reduction of misunderstandings and errors. Digitally represented resource descriptions (i.e. AMDs/Emplacements and MDs/BPs) will also improve the quality of a production system, firstly by ensuring in advance that all the parts will fit together, and secondly, if the processes can be simulated digitally then it can be verified that the system meets the product requirements. All the necessary information for making these verifications is available in the MD/BP. The following subsections will discuss these cases in detail.

The concept should ease the system integration efforts, and make it more straightforward and simpler. The development of the control system implementation can utilise the capability and the skill definitions, providing the interface descriptions of the different functions. These interfaces (i.e. skills) can be utilised in control system composition and orchestration, in which the skills are connected into the sequences of executable actions, and given the parameter values needed to adjust the manufacture of the product. The use of capabilities and skills is discussed in the next section.

In addition to the controls, the mechanical HW integration will also benefit from the concept. The integration of a system through the mechanical interfaces and the compatibility of connections can be ensured a priori, first, through the check of interface mappings and later with the digital

mock-ups. Tools can be developed to guide the service staff in the system reconfiguration operations. It can visualise and instruct how the modules should be assembled into a system, i.e. which interfaces to use to connect two modules together. Again, the tool would retrieve the necessary information from the MDs, accompanied by a some sort of layout definition. From the end user point of view, the concept will also secure their investments as the re-usability and sustainability are of key interest in the concept and naturally built in.

7.1.2 Utilising the Capabilities

The first way to utilise the capabilities is during the matching of the product requirements to the production resource offerings (See 3.2.2) during the design of the production system. On one hand, the product requirements are presented in terms of capabilities, first by the product designer and later refined by the production engineer. This specification defines which capabilities are expected to appear, in which order, and with which parameters and values, in order to manufacture the product. On the other hand, the production resources' functionalities are defined through the capabilities and the skills. These appear in lower levels of the capability hierarchy. These two extremes are then matched to each other during the design phase of the production system, and resources and resource combinations are found which can provide the requested functions for manufacturing the product. This is discussed more thoroughly in [65], and has its basis in [73, 75].

Secondly, the module capabilities/skills can be used to create the control application for the production system, as illustrated in Figure 7.2. The left side represents the implementation of a production module (M1), illustrated with a green dashed border, and the right side, two different system implementations (S1 and S2) each utilising a different architecture (A1 and A2).

The module M1 contains two capabilities (C1 and C2), which are defined in the BP of the module as described earlier in Ch. 6.2.3.3. The BP further shows that the module provider has implemented these capabilities according to three technological solutions or architectures (A1 to A3). A technological solution means, in this case, a specific implementation of a specific control concept, such as IEC 61131, IEC 61499, Multi-Agent System (MAS) or RESTful WS, using specific communication protocols. In the case of A1, the vendor implements only the C1. This specific implementation of C1 and its interface with technology A1 is denoted as C1.A1. Further, the vendor decides to support another technological solution in the second architecture, A2. In this case, they implement both capabilities denoting these correspondingly, C1.A2 and C2.A2. Finally, the vendor implements only the C2 for A3, denoted by C2.A3. These interface implementations of the capabilities (C1.A1, C1.A2, C2.A2 and C2.A3), following selected technological solutions, are made public for the module users and are at their disposal. In other words, the module users, in their implementations based on a specific architecture, cannot utilise directly the capabilities (C1 and C2), but only the provided architecture implementations. The content of the capabilities (C1 and C2), can be accessed, but additional interpretation and mapping is needed before they can be utilised in an implementation of a new architecture, or used for other purposes.

The module controller executes only a single control implementation, containing internally only one implementation per capability (C1 and C2), but it offers multiple interface channels to access these capability implementations. There is an internal implementation that links the interfaces of different technological solutions to an implementation-related interface. This is hidden from the module user. For example, both the C1.A1 and the C1.A2 are linked to and operate with the very same implementation of the C1 at the module controller (solid orange arrows in Figure 7.2). Each module has its internal implementation of control algorithms and logic, which is linked to the capabilities described in the BP file. This is completely hidden from the BP and module users, and it is secured to the IP of the module vendor. This part is represented with a ribbon at the left edge of the M1.

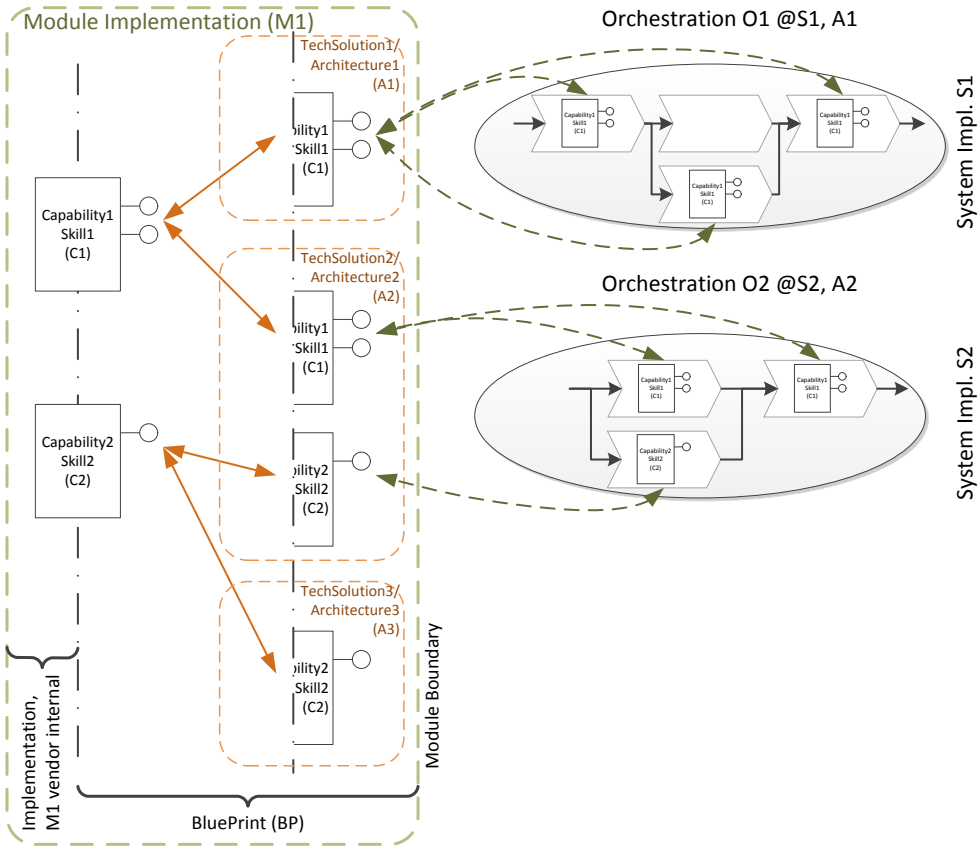


Figure 7.2: Utilisation of capabilities/skills for control application. One production module is used at two different systems following two architectures.

Later, when the system integrator is implementing the system (S1 and S2), they first choose an architecture to follow. This must reflect the selection made by the module vendor, so that there is a match between these two architectures – the provider (left) meets the needs of the user (right). Figure 7.2 illustrates the case where the same module (M1) is used in two different systems (S1 and S2), both of which use different architectures (A1 and A2). The system is configured from the modules and in the case of control modularity, this means orchestration of the modules and, especially, naming their capabilities in a desired sequence. The orchestration utilises the implemented capability interfaces (e.g. C1.A1) to operate the module.

In the case of system S1 the application follows architecture A1. Thus, the module M1 can be connected and used only through the interface C1.A1, making only the C1 interface available for the S1 (the C2 remains not available). The system orchestration implements, executes, and monitors the execution of the process, and calls on the M1 to do the desired operations according to the process sequencer's commands. The dashed green arrows in the diagram show how the S1 makes three calls to C1.A1 in different process phases. In the case of S2, the application follows A2. Thus, the M1 can be used through the interfaces C1.A2 and C2.A2. The M1 is still the very same module, with the very same implementation from the module vendor, but the interfaces which S2 uses to access M1's capabilities are different. In the illustration of the process for S2, C1.A2 is called twice, while the C2.A2 is called once in parallel to the first call of C1.A2.

The system implementation and process orchestration, which are marked with E and F in the framework, Figure 5.10 (p.98), are beyond the scope of this thesis.

7.1.3 Utilising Module's Kinematic Model

The kinematic model can be utilized, for example, during off-line programming, system simulation and for on-line controls. The applicability of the kinematic model is illustrated in Figure 7.3, which represents the two different kinematic positions of the TUT μFactory gantry type manipulator module as a wire frame model, illustrated in Figure 6.6 (p.118). The two positions are: (a) the minimum or home position, and (b) the maximum position.

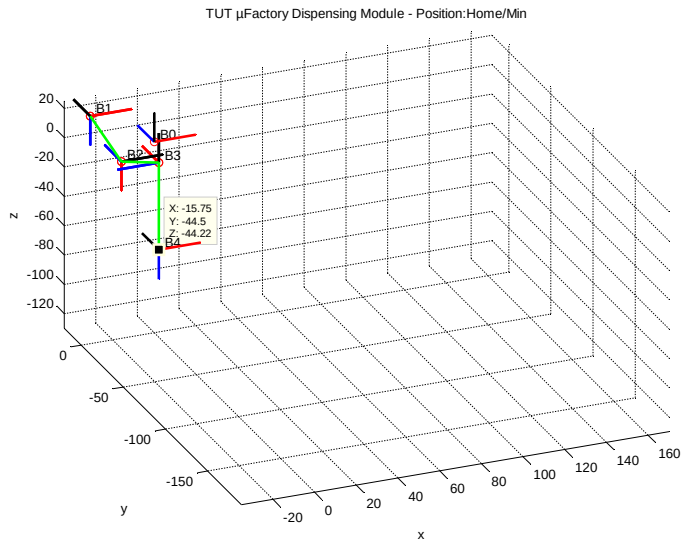
A few different coordinate frames are defined: Interface Frames (F_i), Body Frames (B_i), local origin of module (O_L), and Global Frame (G). F_is are poses of interface mounting points and interface ports to other modules. B_is are origins of moving bodies. O_L is the module's local origin and G is a production cell's global origin.

The origin of the coordinate system is located in the module origin O_L in Figure 7.3, and it overlaps with B_0, which is at the same time the interface, denoted as F_{11}, used to connect this module to its preceding module. There is a constant transformation of $\overrightarrow{B_0B_1}$ represented by the first row in the DH parameter set (Table 6.2 (p.118)). Next, $\overrightarrow{B_1B_2}$ is defined by the second row and the actual position on the Y-axis, $\overrightarrow{B_2B_3}$ is defined by the third row and the actual position on the X-axis, and so on. Finally, after processing all the DH parameter rows, the pose of the manipulator tool's mounting interface, interface frame $F_{21} \equiv B_4$, is defined as $\overrightarrow{B_0B_4}$. The interface frame F_{21} is rotated -90° around B_0's X-axis. In case a), where all axes are in their minimum positions, the F_{21} is located to ${}^{B}_{O_L}\mathbf{r}_{\mathbf{a}F_{21}} = (-15.75, -44.5, -44.22) \equiv P_{21a}$, as printed in the yellow box beside B_4. Respectively, in case b), all axes are in their maximum positions, then the F_{21} is located to ${}^{B}_{O_L}\mathbf{r}_{\mathbf{b}F_{21}} = (144.2, -172.5, -116.2) \equiv P_{21b}$. The manipulator can reach any spatial position between these two extremes by just selecting respective axis positions for the X, Y and Z -axes from the manipulator's controller.

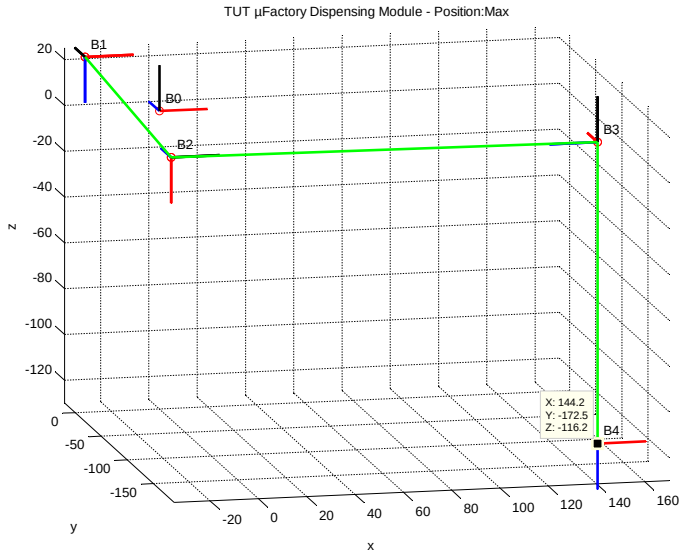
7.1.4 Connecting Modules through the Interfaces

The interfaces are one of the most important aspects the BP document defines. The interfaces determine which modules can be connected together, and how and where. Two interfaces can be connected when they use the same implementation of the same interface, but have opposite gender, or the gender is defined as neutral in both parties. The third option is that the interfaces contain the same kind of features, allowing mating to take place. These conditions were discussed earlier in Ch. 5.3.3.1 (p.94). In mating the coordinate frames of two interfaces from two separate modules, they are brought together so that they overlap each other both in position and in orientation.

The relation between the two interface frames of a module can be determined with a traverse via the local origin (O_L). For example, this will be of interest when defining the pose of the connecting interface to the next module (F_2) relative to the base mounting interface of the module (F_1). In this case, the relation between the two frames ($\overrightarrow{F_1F_2}$) can be determined by reversing the vector $\overrightarrow{O_LF_1}$ and then advancing the vector $\overrightarrow{O_LF_2}$. The chain can be longer if there are more frames between the two end frames of interest, or there is a kinematic structure in the middle. The pose of a single interface frame can be conducted even relative to the Global Frame (G). This method allows any position and orientation of an interface frame in the series of interconnected modules to be determined. Furthermore, this information can be used for tasks like the study and visualization of the overall layout, a feasibility fit study, off-line programming and collision detection.



(a) Home position



(b) Maximum position

Figure 7.3: Denavit-Hartenberg kinematics model for TUT μFactory Linear Manipulator

Additionally, other connectable interface pairs may exist, but solving those needs additional knowledge. Such cases are discussed below. The following example illustrates a situation and its associated rules. Let us assume that we have an interface IF1. This is an interface with genders and therefore it has male and female parts that are connectable with each other. The IF1:F(emale) represents a mechanical interface with five threaded holes and the IF1:M(ale) plate has five bolts on it. Each bolt is attached to the interface and each of them can be turned separately. Let us further

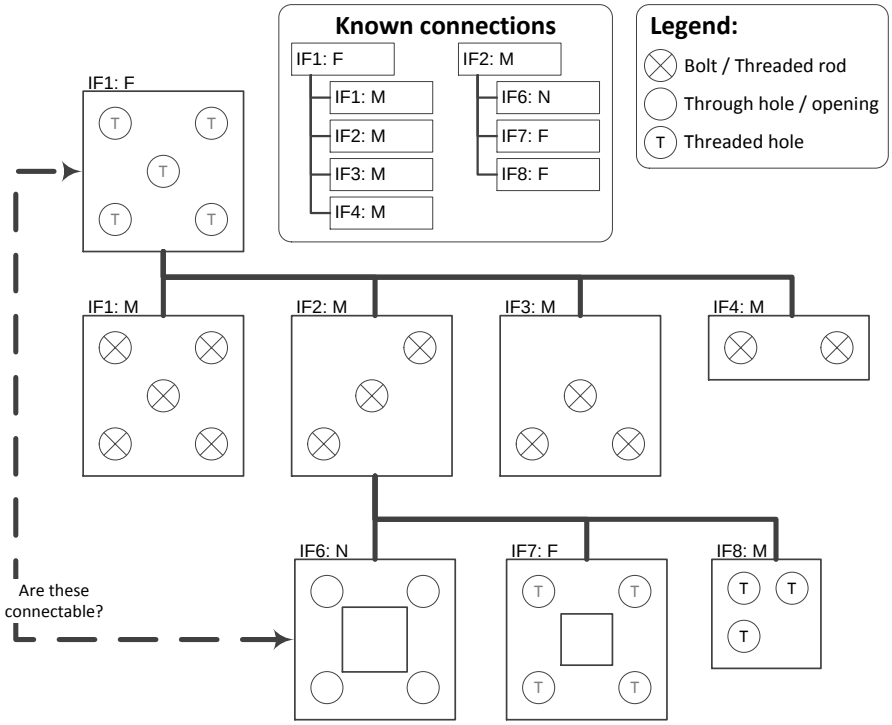


Figure 7.4: Example and issues with interchangeable interface connections.

assume there are other male interfaces, IF2:M, IF3:M, and IF4:M, which all are connectable with IF1:F. The interfaces and their connections are illustrated in Figure 7.4. Similarly, there are IF6:N(eutral), IF7:F, and IF8:F, which are defined to be connectable with IF2:M. The IF6:N is a flange without any threaded holes or bolts, just through holes. Therefore, it can be mounted on IF2:M with two extra nuts fitting to the corner bolts.

The first question can be made: Which interfaces are directly connectable to each other? This is the list given in this example by the assumptions, and visualised by the hierarchical tree structures in the top right corner of Figure 7.4. If no prior knowledge of the interfaces is available, this is hard problem for a computer to solve. If only the IDs of the interfaces are provided, the task is impossible. A small amount of additional classification data will not help much either. Only complete CAD data could provide enough information for computer algorithms to be made for solving the mating interfaces and creating connection networks between the interfaces. However, this would be a laborious and exhausting process.

Even if the interface were mechanically and electrically perfectly connectable, the functionality of the interface would be blocked or even destroyed after connection. Understanding the functional behaviour of the interface makes a computerised evaluation an overwhelmingly weighty operation. If this aspect were covered by the computer, the description would require a bunch of additional specifications of the intended and even sometimes hidden purposes of the interface. As an illustration, we use a simple example with IF2:M and IF7:F in Figure 7.4. If we make a further specification that the rectangular hole at the centre of IF7 is used for a vision system, this functionality breaks the connection between these two interfaces, because the functionality condition is no longer satisfied, i.e. the camera could not see through the plate of the IF2:M. Or, mating could destroy the interface because of a collision between the centre bolt of IF2:M and the lens of the camera, if the camera were close enough to the interface surface.

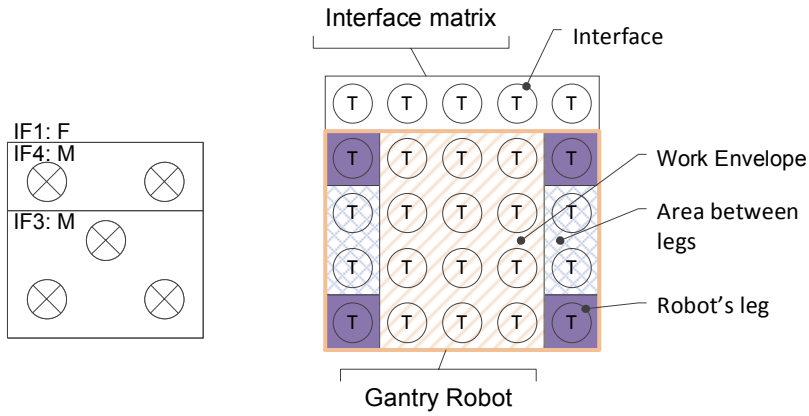


Figure 7.5: Example at left: Split of interface, and right: space reservation by a robot.

The second problem is the cross connections. Let us make further assumptions in our example in Figure 7.4. We have the IF2:M connectable with IF1:F and IF6:N connectable with IF2:M. The question is: Can it thus be deduced that IF6:N is connectable with IF1:F, or not? This is also case-specific and needs an expert to judge it case by case. In our example the IF6:N would be connectable with IF1:F with the aid of an additional two to four bolts. Correspondingly, the IF7:F will not be connectable with the IF1:F in any circumstances, because both have threaded holes for mating. The only possibility would be to use an additional adapter module between the IF1:F and IF7:F that would offer both counterpart interfaces, or some other mechanism, and thus connect these modules together.

Therefore, because the interfaces are designed by human engineers and specified in textual standards, the proposal is that human experts be used for creating the formal libraries of connectable interfaces, in addition to a categorised interface model. The interface designer already has the interconnection of selected interfaces in mind when they specify a new interface, therefore this information could be explicitly stored in 'interconnection libraries'. The chaining of interfaces is not directly possible in the case of merging interconnection libraries, without additional knowledge of the specific interfaces, even if connection paths exist through joint interfaces (like IF2:M in our example). However such joint interfaces are important links, pointing out the interfaces that potentially can be denoted as connectable, but this requires additional evaluation and judgement from a human expert. The interface connection libraries could be implemented as a list containing pairs of connectable interfaces, or as an ontology-based definition. However, a library of connectable interfaces is associated with system design and system layout and is thus beyond the scope of this thesis.

Yet another level of problems arise and need to be considered when the system layout is made. There are additional spatial requirements and cross-linking implications as different production modules impose their own spatial requirements, which is illustrated in Figure 7.5. These concerns are especially valid for matrix-type, repeating interfaces, like an optical table or a Lego block. In this case, several modules are connected to a matrix-type base interface and each module occupies individual interface unit(s). The spatial volume of the inserted module may reserve other interface units, or block other modules from being connected in some, otherwise free, unit locations of the matrix interface.

- **Space reservation**. An interface is a composition of a large matrix of e.g. holes, or there is a larger matrix pattern of an elementary interface unit. Let us have, e.g. a 5×5 matrix for interface A. When placing a 5×4 unit gantry robot on top of it, the robot will be mounted

on the base with two legs, each of which occupy 1×4 units, and which are 3 units apart. This defines the robot's workspace (3×4 units). Each leg has two units free at the centre of the leg, but the space is limited by the construction of the robot. How and under what conditions are these spaces (workspace and holes under each two legs) available for use by other modules? (See shaded areas in Figure 7.5: right)

- **Orientation of an interface**. The interface can be in several different orientations. For example, IF3:M or IF4:M connected to IF1:F can happen in four different orientations (See Figure 7.4: left).
- **Split of interface**. Is it possible to have an interface connected with two or more other interfaces? The simplest answer would be to use 1-to-1 interfaces, and this is certainly the recommendation or rule from this author. 1-to-N type of connections becomes difficult and complex to handle. E.g., on IF1:F, two parallel modules are connected with IF4:M (See Figure 7.4 and Figure 7.5: left). This would be an example of a 1-to-2 connection. The space reservation might be simultaneously an issue, as in the first case, where a matrix interface A is defined and a device consumes e.g. 3×4 interface units. Even if some points under the device were free and accessible, they are not available.
- **(Mutual) exclusion because of orientation**. Depending on the orientation of one connection, other connections are excluded.

3D system simulation and spatial collision detection would be the best methods for solving these issues, but would require the involvement of a human expert. This problem is related to the system layout and is therefore beyond the scope of this thesis. The interface description-related matters are left for future work.

7.1.5 Cell Model, Module Chains and Tool Position

Figure 7.6 illustrates how a system, such as a production cell, is composed of modules by use of the interface information and kinematics models from the BP files. The modules are combined as a chain of interconnected interfaces. The case demonstrates the TUT μFactory cell for dispensing (the right-most cell in Figure 8.1c (p.158)). The cell is composed of four modules: TUT μFactory base module (M1), 3DOF linear manipulator module (M2), dispensing gripper (M3), and controller module (M4). All the interfaces of these modules are represented in Figure 7.6. There are three mating points between these four modules: Connection from M1 to M2, from M2 to M3, and from M2 to M4. These are represented by A to C respectively, circled in green. The A represents the conjunction of interfaces M1:IF03-M-1 (Process interface to next module of M1) and M2:IF01-F-1 (Mounting interface of manipulator module M2). The B is the conjunction of interfaces M2:IF02-F-1 (Tool interface of the manipulator module M2) and M3:IF01-M-1 (Mounting interface of the gripper module M3). The C is the conjunction of interfaces M2:IF04-M-1 (Process interface to next module of M2) and M4:IF01-F-1 (Mounting interface of controller module M4). The joints of manipulator module (M2) (i.e. the kinematic model expressed with DH parameters) are hidden in Figure 7.6, and only the end frame of the manipulator mechanism (M2:IF02-F-1) is shown for the sake of clarity. All the information used for this example originates from the four BP descriptions, each representing its corresponding module type.

The G of the production cell is bound to O_L of the base module M1, namely to the interface frame M1:IF02-F-1 $= F_{1.02}$[1]. The end tip of the dispensing needle (i.e.TCP) is represented by the frame M3:IF02-M-1 $= F_{3.02}$, where the Z-axis of the frame (blue axis) is pointing out from the needle. With this model, connecting these four modules together, it is possible to calculate the

[1] Frame F's index, which is composed as follows. The first digit of the index denotes the module number, followed by a full stop. The next two digits denote the interface id number.

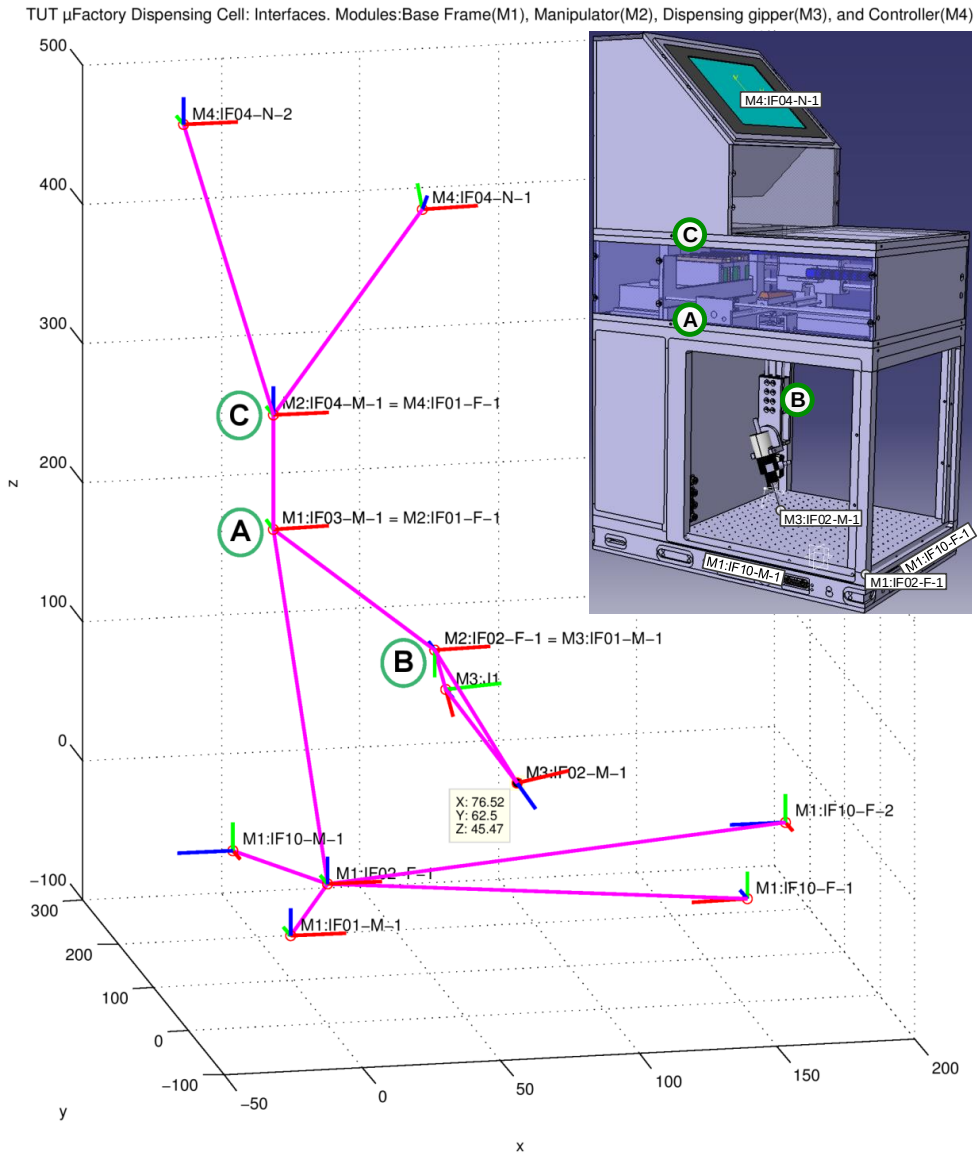


Figure 7.6: Example of layout and assembly of modules via interfaces. Includes kinematic model of the cell. All units are in millimetres. Illustrated cell: TUT μFactory dispensing cell, composed from base frame (M1), 3DOF cartesian manipulator (M2), dispensing gripper (M3), and controller (M4). The label represents the position of TCP relative to the origin of the base frame (M1:IF02-F-1). The circled letters (A to C) denote the joining points, where interfaces from two different modules are united.

position and orientation of the dispensing needle ($F_{3.02}$) by using a forward kinematics model of the entire cell. The $\overrightarrow{GF_{3.02}}$ represents the position of the end effector through the modules and the kinematic chain $G = O_1 = F_{1.02} \rightarrow F_{1.03} \Leftrightarrow F_{2.01} \rightarrow O_2 \rightarrow F_{2.02} \Leftrightarrow F_{3.01} \rightarrow O_3 \rightarrow F_{3.02}$. In Figure 7.6 the manipulator's physical axes are 'driven' in positions (60, 60, 60), leading to the ${}^{G}_{F_{1.02}}\mathbf{r}_{F_{3.02}} = (76.52, 62.5, 45.47)$ (yellow box).

The cell model becomes especially useful when it is appended to the model of the end product (or part for it) and its fixtures. These are defined with a chain of vectors from G using the available interfaces at the base module (M1). This will allow us to calculate the positions and paths for the manipulator by use of an inverse kinematic model. In other words, this all provides the required information for the off-line programming, simulation, and verification of the work cycle. For example, the position of the dispensing needle is defined relative to the product, with an included process gap for the dispensing process. The kinematic chain from the product through G to the TCP of the needle is to be solved. The only free variables in this case will be the positions of the manipulator's axes, which are further solved and stored for the control program. With the same method, other target positions and paths can be solved and stored for the control program for the manipulator. For example, the picking point of a component in the feeder, the component's assembly point in the end product, and the path from the picking point to the assembly point.

7.2 Data Exports and Transformations

The description files can be used directly by experts and various tools which innately understand the format, but another important use case would be how the other tools or definitions would utilise these definitions. However, before any other tools can understand the content of the description files, these need to be either imported by the target tool or framework, or alternatively, the content of the description file is to be exported to some other format that is known by the target tool. Especially the latter alternative would greatly increase the acceptance, applicability and utilisation of the developed description file formats.

The process for exporting the format to another is called, in this case, transformation, as the information expressed in one format is transformed or mapped into another by applying specific rules. The transformation can be performed with different methods starting from manual operations to several automated ones.

The different applicable methods for performing the transformations are discussed next, and some practical examples are provided, demonstrating the potential and possibilities offered by the transformations. One aspect of this is to use these case studies to verify that it is possible to transfer and utilise the modelled knowledge to other formats and environments.

7.2.1 Methods for Exports and Transformations

There are a few alternative methods for implementing the transformations within this context. The manual, pick and place generation of output is left out as a method, because it is very time-consuming and error prone. Therefore, only three automated methods are considered. One is to hard-code the transformation, the second is the use of eXtensible Stylesheet Language Transformation (XSLT), and the third is a mixture of the two.

The hard code method is to use general purpose, object-oriented programming languages (like Java, Python, C/C++/C#). The input file is processed according to the programmed statements and models, and an output is produced. A practical implementation could include the following steps: read the input file into Document Object Model (DOM) tree; perform procedures for transformation, such as search for specific nodes, iterative loops, the merging of data from several

sources, etc., and then, generate the output in the target format. The output can also be processed and presented as another DOM tree depending on the target format used.

This method offers the most flexible and dexterous processing; multiple sources are supported, and if the processing rules get very difficult and complex, this is the preferred method. The drawbacks are the rigidity of the implementation, i.e. the implementation needs more work and effort as the supporting processes (like reading the file, finding right nodes, etc.) require quite a lot of coding; and there is considerable room for errors to occur. Also, the response to changes (like input or output format changes, or changes in processing rules) is weaker as these always require re-coding and testing, which need to be performed by specialised software developer(s).

The XSLT method is a domain-specific language which is a declarative programming method. The XSLT [143] offers a document-processing method which can be used to make a transformation from an XML document to another document. The transformation processing instructions or rules are specified in an XSLT document, which defines how matching nodes in the input file are processed, and what is the output in each case. (See Ch. 6.1 (p.105) and Figure 6.1)

This method offers flexibility for processing. The main advantage is that the data modelling expert can make the processing instructions by themselves into an XSLT document by using this domain-specific language. There is no need to utilise another expert for programming the SW application. The process is fairly stable and standardised, and there are several XSLT processors available as open source and freeware. This way, the amount of effort to make the transformation and the potential places for errors are reduced to the minimum, because the transformation is specified directly with the domain-specific language, shortening and reducing the phases to implementation. The cost of implementing the transformation is lowest and the quality is potentially highest.

There are some drawbacks. One is the need to learn the XSLT language, although the learning curve shouldn't be too steep because the language is based on XML and XPath [146]. Another limitation of this method is that only one input file and transformation template can be used at a time, so that if the transformation is more complex and requires use of several input and/or XSLT files at a time, this method will not be suitable by itself. The XSLT processor is capable of performing only the same transformation process (i.e. the same XSLT instructions) several times in a row to other input files to produce a set of transformation pairs of input and output files. However, scripting, for example, can easily be used to batch process several files, (still one at the time though), to make a transformation to all of them at once. In such a case, however, the transformation itself gets very complex and fancy, which is a point where this method may have reached its limits. However, this limit did not occur in the presented case examples.

The mixed method combines hard coding and the XSLT processing methods. In this method, coding (or scripting) is used to add one organisational layer above the XSLT method. This combines the best parts of both the previous methods, and overcomes the limitation of XSLT only being capable of processing one input and one XSLT file at a time. In this method, a set of XSLT files are used to produce output fragments, which are then used by the organisational process. A traditionally programmed application (hard coded/scripting) provides the selection of both the input and the XSLT file, executes the transformation process, and captures the output. The hard-coded application then merges the output with another output, or it can pass on the output as the input for another transformation process.

This method makes use of tested and proven XSLT processing as far as possible and merges into it the flexibility and freedom of the hard-coded application. The former offers a proven platform, and reduces effort and possible errors. The latter can easily handle the issue of using several sources and it can perform merging and sequencing operations for the final output. The

drawbacks of this method are that it requires more effort and craftsmanship to make the hard-coded part live and integrated into the overall process. The data modelling expert can still make the required core transformation process instructions, and they can even be changed at later phases, if, e.g., the output merging process is not affected. The overall process and application need to be specified in more detail and normally more parties are involved for the implementation.

Conclusion of Transformation Methods The transformation implementations related to this thesis utilise the two latter methods, and the XSLT method is used as much as possible. The XSLT specification version 1.0 is used, because it is better supported by the tools and libraries used by the implementations.

7.2.2 Human Readable Documentation

One of the first intended uses for the XSLT was to create an HTML document out of an XML document. This original intention is also applied in this case - Documentation of the description files as HTML pages generated directly from the sources.

The main objective is to provide human-readable documentation of the Emplacement and the BP description files. The documentation is offered to the user in the HTML format which can be viewed with a standard web browser, and it is generated on the fly by the XSLT transformation process. The set of XSLT files are created to contain the transformation rules for each represented description format and version. During the transformation, the document is enriched, formatted and clarified for the benefit of the reader. Still, the original knowledge source remains simple, condensed, and allows the data to be defined only in one single place, and referenced in others by internal document references.

The **enriching** means, for example: addition of summarising elements, opening the references and links, and creating hyperlinks in the text. Summarising elements are, for example, tables of contents and listings of available or extended $\text{Profile}_{\text{Empl}}$s. These elements open the document structure for the reader (table of contents) or collect all available options into one place (list of implementable $\text{Profile}_{\text{Empl}}$s). Often, the summarising element is enriched with hyperlinks supplying a link to another place in the document. This linking is also created during the transformation by generating the target labels and the hyperlinks to these from other parts of the documentation. Opening the references and the links in the original XML document means, for example, creating hyperlinks from the links and references within the XML document, visualising images in the output document together with their captions, and adding the titles. These are illustrated in Figure 7.7.

Formatting means, for example: representing listings of elements as tables, adding headings, sorting, and applying Cascading Style Sheets (CSS) for visual effects. Different element listings things like properties, interfaces, skills, etc. can be visualised as tables. During the transformation, first a heading row is generated, then the content of the listing is sorted and filtered, and finally the output is generated in a table format. Sorting can also be used in various other places to, e.g., sort $\text{Profile}_{\text{Empl}}$s in alphabetical order. The CSS style sheet is applied to the whole output document. This is standard process for HTML pages and is not directly involved in the transformation process. However, this technique is used to provide visualisation for the documentation by adding colours, margins, borders, and paddings, by formatting headings and other selected HTML elements, by creating a format for representing the images and their captions, and so on. See Figure 7.7 and Figure 7.8.

Clarification includes, for example: following the references and opening containing data, and revealing the concept of 'Extension'. These are illustrated in Figure 7.8. The references are followed and the referenced data is presented in extended form in the place where it was referred to. This way, the same information appears in various places in the viewed document,

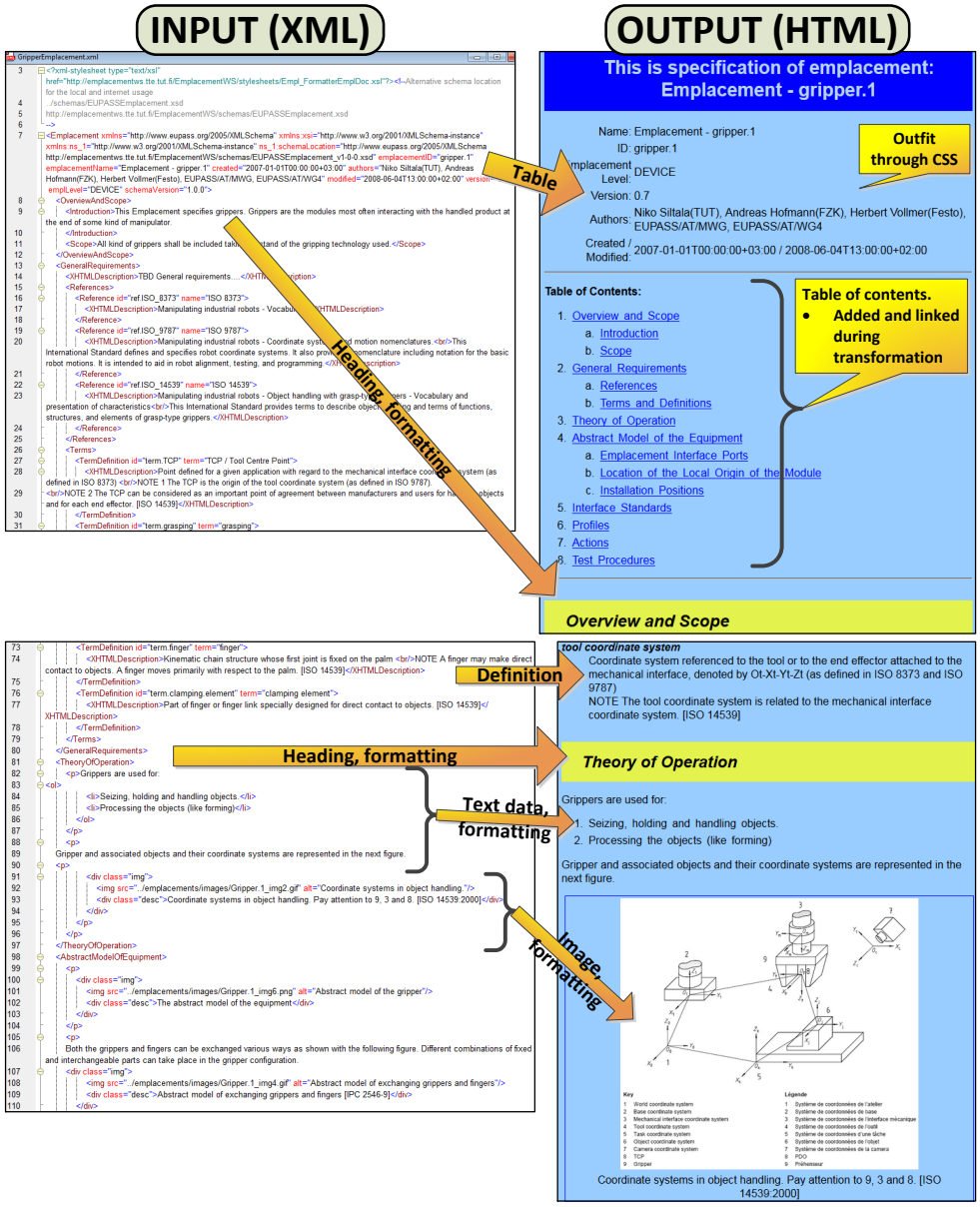


Figure 7.7: Generating HTML documentation from XML file through XSLT transformation: Table formation, adding table of contents (i.e. summarising content), hyperlinks and visualising images.

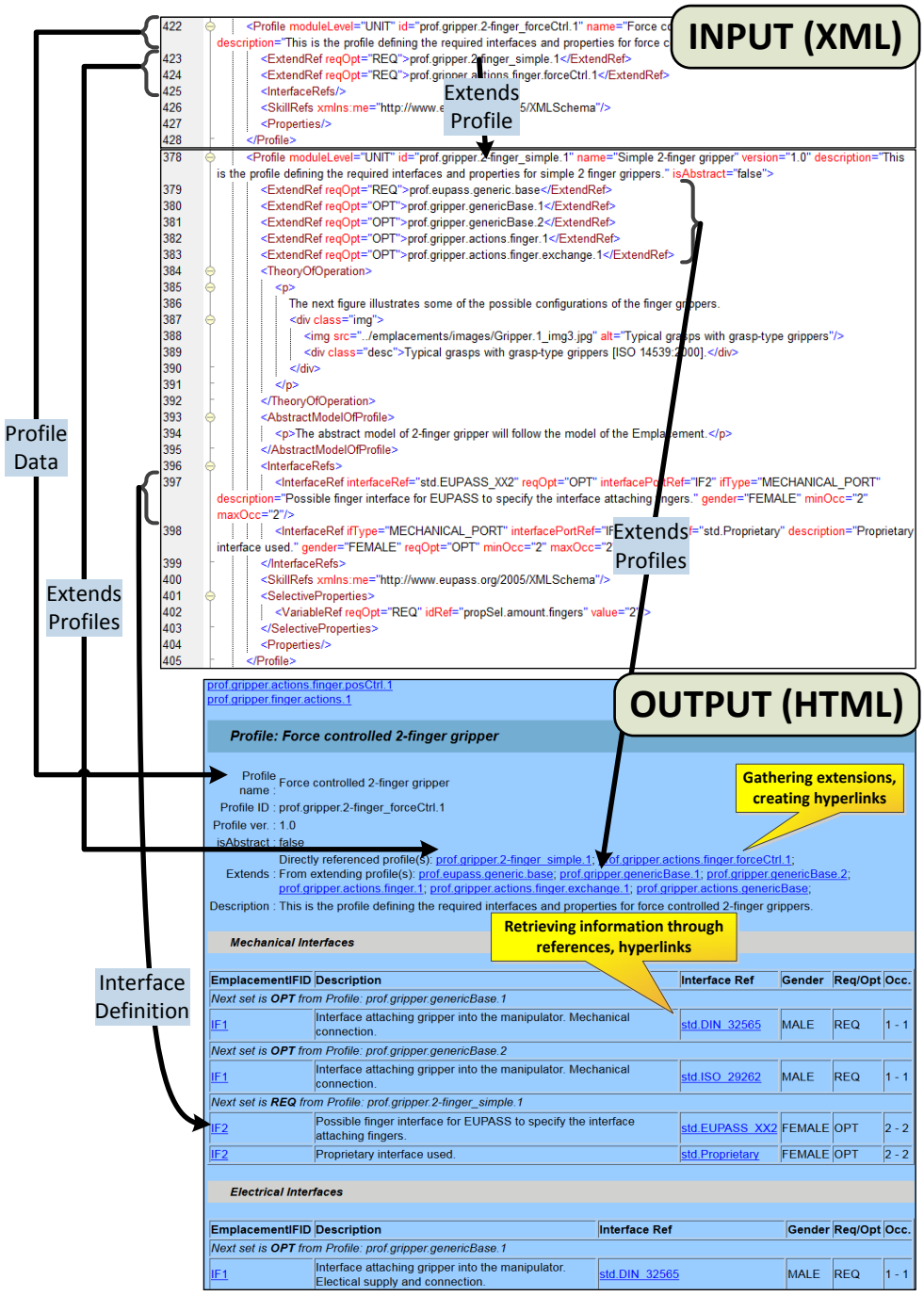


Figure 7.8: Generating HTML documentation from XML file through XSLT transformation: opening references and extensions. Example of table formatting.

but it increases the readability as the user does not need to jump from one place to another while reading the document. This arrangement assists the user to more easily get an overall view of an entity, like, for example, all the available interfaces for a $\text{Profile}_{\text{Empl}}$. The arrangement of using referencing and the concept of extension allows the information to only be defined in a single place (in e.g. the Emplacement or BP document), which increases the maintainability and consistency of the specifications. This choice makes the use of files, for example the visualisation, a bit more complex. Referencing applies to objects such as standards, interfaces, variables, properties, etc., while the concept of extension applies to $\text{Profile}_{\text{Empl}}$s, skills, etc.

However, if the referencing and the concept of extension create an issue for the utilising application, a specific XSLT transformation can be created that opens and flattens the specification. This would include that extended, e.g. $\text{Profile}_{\text{Empl}}$ descriptions, are merged and completely revealed in each place they are extended. This increases the size of the file substantially as the same information is repeated in several places. The output file remains valid with the defined data models (XSDs), and it corresponds to the original file from the specification side, but would just be a bit easier to use.

The XSLT file generating the human-readable documentation for Emplacement is illustrated in Appendix E.1, and the resulting output is represented in Figure 7.7 and Figure 7.8. The EmplWS can be used to generate and view the documentation on-line (See Ch. 7.3.1 (p.143)). Table 7.1 summarises the order of magnitude of the different XSLT files used to generate the human-readable documentation. The first one, on row one, is for generating the human-readable documentation of an Emplacement file, and the second is for the BP description.

Table 7.1: Order of magnitude of XSLT files generating the human readable documentation

Purpose	**Filename**	**Lines of code**	**Size (KB)**
Emplacement	Empl_FormatterEmplDoc_v1-0-0.xsl	1641	51,1
BP	BP_FormatterBPDoc_v1-0-0.xsl	1772	53,0

7.2.3 From Emplacement to BluePrint Skeleton

The module provider needs to make the BP description defining their production module(s). The new module requirements, design process, and the impact on the module provider's side are discussed in more detail in Ch. 7.1.1 (p.125) and in Figure 4.2 (p.64).

In addition to the semantics defined by the BP XSD, the BP specification needs to comply, with the content of the specification of the selected Emplacement and $\text{Profile}_{\text{Empl}}$ which the BP finally implements. In order to help the module provider in this process, and to provide a valid, well-formed document with the correct content as a starting point, a transformation procedure from the Emplacement to BP is defined, implemented, tested and analysed. This procedure produces a '*BP Skeleton*' file, which is a base for the final BP file that the module provider prepares. It contains in right place, all the applicable information from the $\text{Profile}_{\text{Empl}}$, such as the interfaces and the skills, and all the variables and properties. It is appended with empty placeholders and structures to start with, so the module provide can just start filling in or removing the information according to the final needs of the production module. All mandatory structures are in place, and the file is semantically and contextually valid. So it can be directly validated against the BP schema (XSD), and it will pass the XML tests for being both well-formed and schematically valid. This operation will be of great benefit to the module provider, and will save a lot of time and effort in cases when there already exists a suitable $\text{Profile}_{\text{Empl}}$ definition.

The implementation of this transformation utilises the mixed method. In this case, the main job is done by the XSLT processing, but a Java code is used to select the desired input file i.e. the

Emplacement, and more importantly, to modify the XSLT document somewhat before it is passed to the XSLT processor. The modification is not great, it is just to assign the right $\text{Profile}_{\text{Empl}}$ ID to XSLT variable 'var.profile', so that the selected $\text{Profile}_{\text{Empl}}$ can be processed to the output. After this small modification, standard XSLT processing produces the requested output of a *BP Skeleton*.

The XSLT file (*Empl_FormatterBPSkeleton_v1-0-0.xsl*) for producing the *BP Skeleton* from an Emplacement file has 682 lines of code and is only 28,7 KB. The XSLT is illustrated in Appendix E.2. The EmplWS can be used to generate the specific *BP Skeleton* on-line, according to the Emplacement and $\text{Profile}_{\text{Empl}}$ selections made by the user (See Ch. 7.3.1 (p.143)).

7.2.4 From Module Description to Other Formats

The transformations can be used not only within the proposed concept, but also for transforming the module information into other description concepts i.e. exporting information. However, there the transformation may not be that easy, clear, matching, complete and comprehensive, and some information is likely to be lost. This is because of the use of different knowledge models and modelling concepts. For example, different terminology and units of measurement are not an issue if, and only if, the underlying knowledge models are compliant with each other.

This kind of transformation is studied with the transformation of a BP file into an AutomationML file fragment representing a production module. AutomationML is discussed in Ch. 3.4.1.1 (p.37). The objective of this exercise would be to find out whether the transformation from BP to AutomationML is possible. What are the limitations of such a transformation? What information can be transferred and what will be lost? What sections need some extra work, and what parts does the BP lack which are required by AutomationML? However interesting this scenario is, it must be left here for future work.

7.2.5 Other Usage

In addition to the previously listed transformations and exports, some other kind of transformations are created and utilised. Inside the EmplWS, a set of small and fairly simple XSLT transformations are used extensively. One example is to create a comma-separated listing of $\text{Profile}_{\text{Empl}}$ names from a single Emplacement file. Another example is the creation of a listing of $\text{Profile}_{\text{Empl}}$ IDs with some extra information like descriptions, and containing Emplacement file ID information. The results of both these examples are utilised inside the EmplWS, as part of the web page's information content and visualisations. Table 7.2 visualises the magnitudes of different XSLT files used for these internal transformations.

Another use for the XSLT transformations is to provide description file upgrades from an earlier version to the next. The model of the description (i.e. the XSD) is also changed when the description version changes, after which the schematic validation fails for the earlier files, which is the expected behaviour. In order to ease the move from one version to another, a transformation can be defined and utilised. It produces a new description file, which will be according to the new XSD version and will be semantically valid. A data modelling expert just needs to create a new XSLT file, and define and fill in the rules used to alter or append the content of the earlier description to correspond with the intentions of the updated information coming from the new specification. This will ease the job, as a great part of the process can be automated and errors in repetitive parts avoided. Some changes can even be completed by the automated process, such as altering the name of some entity, adding or updating some common information like the version number or Uniform Resource Locator (URL). This kind of manually-triggered activity can be implemented as batch processing of a folder of files at once, with the help of scripting and an XSLT for each upgraded specification. Of course, this process cannot provide any new data for

the description, and in such cases, a human expert needs to post-process manually the descriptions. Below, part of Table 7.2 represents the characteristics of some files for this purpose. The file size is affected by the amount and complexity of the changes between the different XSD versions.

Table 7.2: Order of magnitude of the XSLT files used for other usage

Purpose	Filename	Lines of code	Size (KB)
List of $Profile_{Empl}$ names	Empl_FormatterProfileNamesOf Empl.xsl	18	0,80
List of $Profile_{Empl}$ IDs with desc	Empl_FormatterMsgGetProfile IDs.xsl	41	1,54
Version update: Emplacement v0.9 ⇒ v1.0.0	Empl_transition_v0-9_TO_v1-0-0.xsl	72	2,50
Version update: BP v0.9 ⇒ v1.0.0	BP_transition_v0-9_TO_v1-0-0.xsl	130	4,98

7.3 Tools

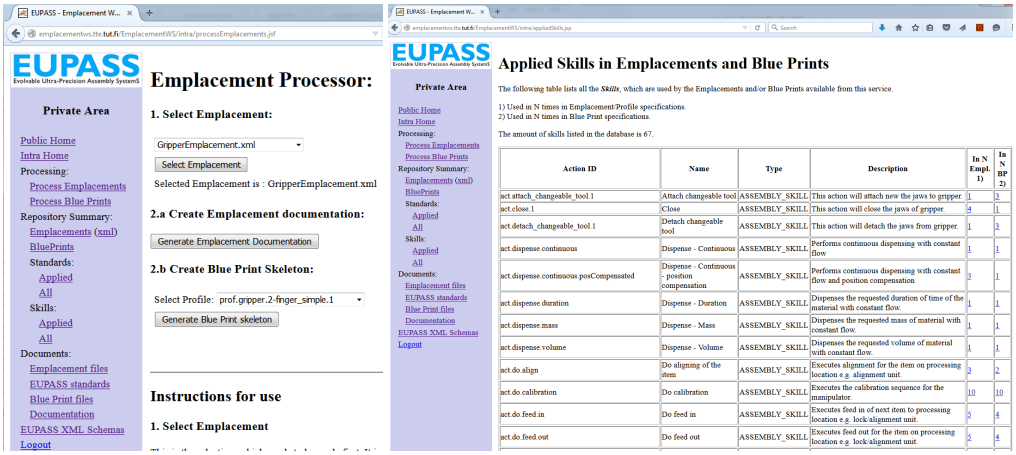
A set of tools are developed and used to create, edit and modify, visualise, and process the Emplacement Concept files. As a starting point, a generic XML editors and Integrated Development Environments (IDEs) are used to develop the models (XSDs), edit the files and process (XSLTs) them in various forms. *Altova's XML Spy* [176] is one such generic XML editor that has been extensively utilised during the development process. Later, more specific and specialised tools associated with the Emplacement Concept are introduced, and these are discussed below.

7.3.1 Emplacement Web Service

The Emplacement Web Service (EmplWS) [173] is a web-based SW tool, implemented with Java and Java Server Pages (JSPs), for managing the Emplacement and the BP files. It is used for distributing information about the production modules, and provides a market place for the module providers. It can make some processing actions, such as generating HTML documentation about the Emplacement or BP, or generating a BP Skeleton document for a new production module (See Figure 7.9a). The interface standard and process ID-related information are included.

The EmplWS can be used as a design support tool. This can be utilised to search for Emplacements or production modules implementing a specific skill (See Figure 7.9b) or a specific interface (See Figure 7.10). The views for applied standards or skills list all interface standards and skills defined in any of the Emplacement or BP documents, stored in the EmplWS DB. The second last column has the number of Emplacements implementing a specific Interface (IF) or skill, and it provides a link to another page, which shows details about the Emplacements and $Profile_{Empl}$s implementing the IF/skill. The last column has the number of BPs implementing the IF or skill, and behind the link a page listing the details of these BPs opens. These use cases are illustrated in Figure 7.10. The EmplWS is discussed in more detail in [126].

Figure 7.7 and Figure 7.8 show a view of the documentation-related outputs from this tool. In addition, the tool offers a WSDL interface for other SW applications to connect and utilise the provided services. The EmplWS is developed and maintained by the author.



(a) View for Emplacement processing

(b) View listing applied skills in EmplWS

Figure 7.9: Views from EmplWS [173]

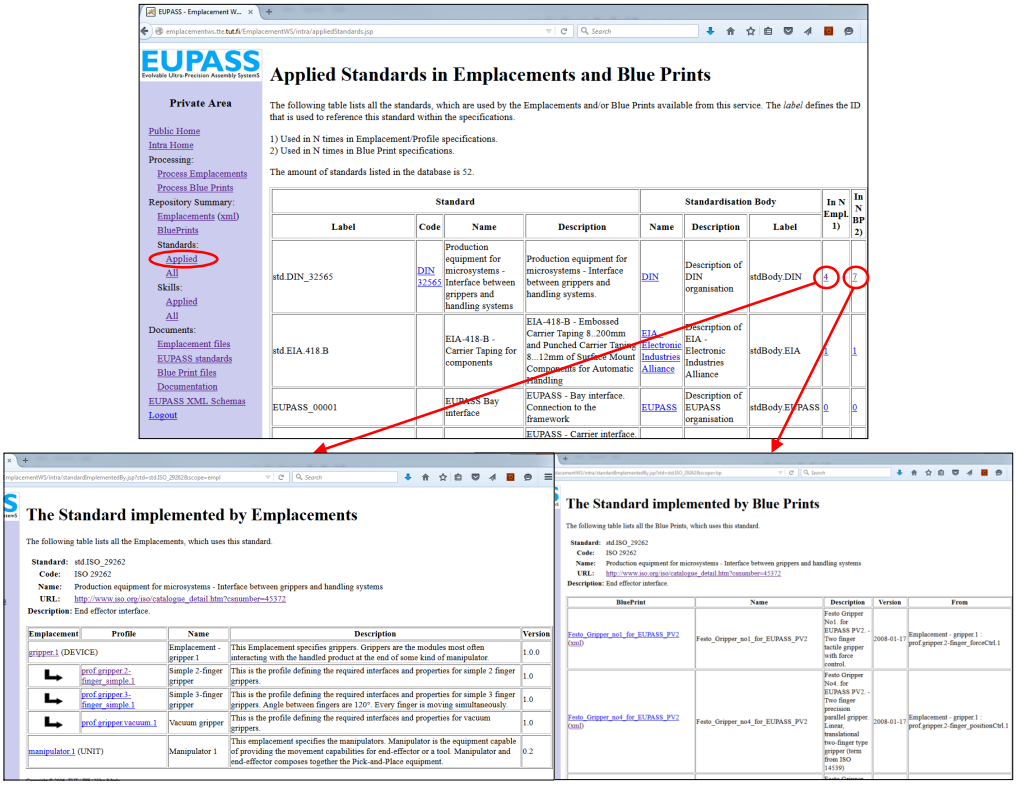


Figure 7.10: Views summarising information about applied interface standards [173]

7.3.2 Blueprint Editor

The Blueprint editor is a stand-alone SW application providing a GUI to browse, view, edit, and create BP files. It utilises the EmplWS as a data source through the server's WSDL interface. After the user has selected the desired Emplacement and $\text{Profile}_{\text{Empl}}$ combination, the BP Editor is capable of retrieving customised BP templates (BP Skeletons) from the server. This tool has been developed by Pedro Mendes, UNINOVA [130]. A new BP editor tool is also under development at Tampere University of Technology (TUT).

Figure 7.11 illustrates two different BP files, totally, in four different screenshots. Top left is the view of a business property and top right shows the interface standard details for a gripper BP. Bottom left is a view of the module's general information, together with its physical properties, and bottom right is an instance of the mechanical interface with DH parameters for a linear manipulator module BP. The latter is the same module as the one used in the kinematics example (See Figure 6.6).

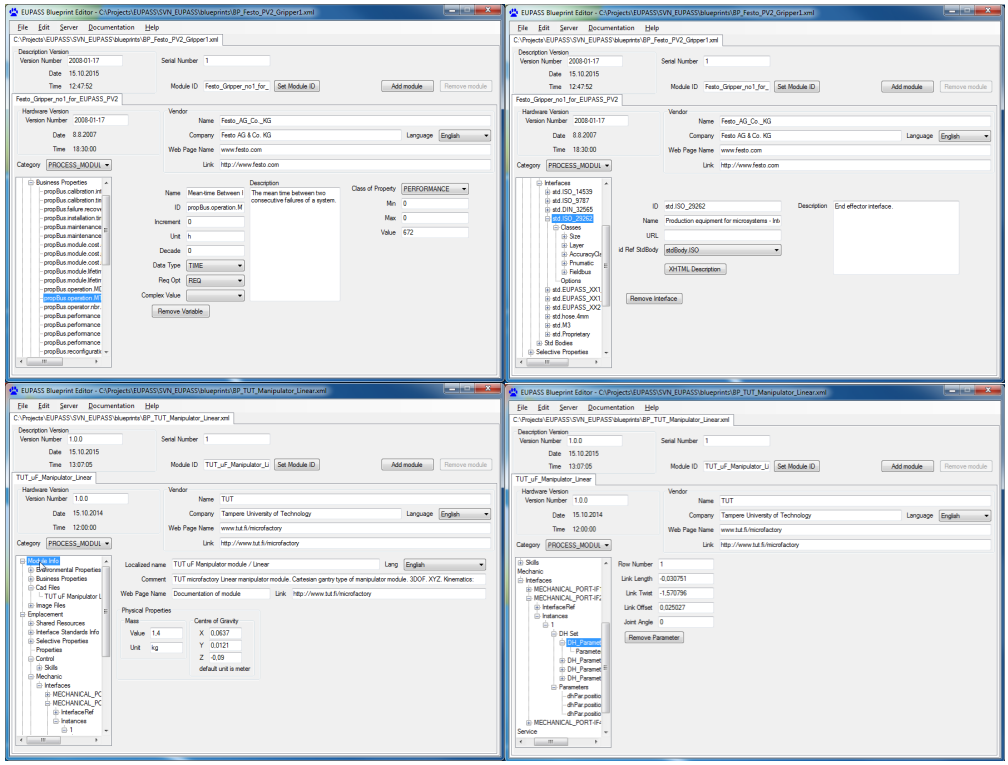
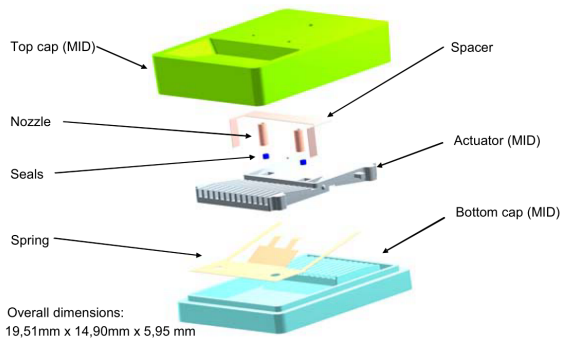


Figure 7.11: Four screenshots of the BP Editor, representing different views of two BP files

7.3.3 Other Tools

Other parties have made a few software applications supporting the concept. These are mainly able to read the descriptions, and extract from there the essential information fragments that the specific tool utilises. Below are the intended use-case scenarios and how the different descriptions are utilised.

- UNOTT's Assembly System Configurator. It is a production system design-related SW. [34–36]



(a) Miniature valve

(b) Doorhandle for a washing machine

Figure 7.12: Assembled end products for CaseEnv 1 [31]

- ITIA's tool for evaluating Assembly system performance, basing on discrete event simulation
- Beckhoff's control system for linking modules together and automatic code generator for PLCs.
- ITIA's tool for assisting visually assembly system HW configuration.

7.4 Implementation of Case 1

The EUPASS Platform Version 2 (PV2) environment (CaseEnv 1) was built during EUPASS project at Fachhochschule Nordwestschweiz (FHNW) [166] Switzerland in the autumn of 2008. The environment is used to demonstrate the multi-discipline development results of the EU FP6 EUPASS project. The main areas of focus and the results relate to the system design concept, modular assembly architecture and interfaces, process technologies, and the communication and controls. The part of the PV2, which is considered as CaseEnv 1, is a modular assembly line consisting of three assembly cells, one inter-station conveyor and two end lifts - constituting a total of 12 modules [31]. The first of the assembly cells is a manual station, the next is semi-automatic, and the third is a fully automatic one. [27–30]

Figure 7.12 illustrates the two case-products which are assembled by CaseEnv 1. The first one is a miniature valve and the second is a doorhandle for a washing machine. The same assembly system is used to assemble both of the case products. After changing the modules and reconfiguring the set-up and control software, new carriers containing the next product can be sent in. The changeover takes place fairly quickly and the line is back in operation in a few minutes.

Figure 7.13 represents the liaison graph and available modules for CaseEnv 1 in the configuration for assembling the miniature valve. The production line is composed of three cell frames (2, 3.a and 3.b) and an auxiliary pallet-circulation system, including two end lifts (1.a and 1.b), the inter-station conveyor (4), and three sets of intra cell conveyors (5.a to 5.c and 6.a to 6.c). The first automatic cell (3.a) performs a pick and place operation, and it consists of: a) one 2DOF manipulator (8) with finger gripper (11), and b) base for feeders (13) with one tape feeder (14). The second automatic cell (3.b) performs vision-aligned dispensing and assembly operations. It consists of: a) one 4DOF manipulator (7) with finger gripper (10); b) one 1DOF manipulator (9) moving; c) up and down looking vision system (15), and d) sophisticated dispensing module providing a glue-dispensing capability (See Figure 7.14b). In the next subsection, the digital

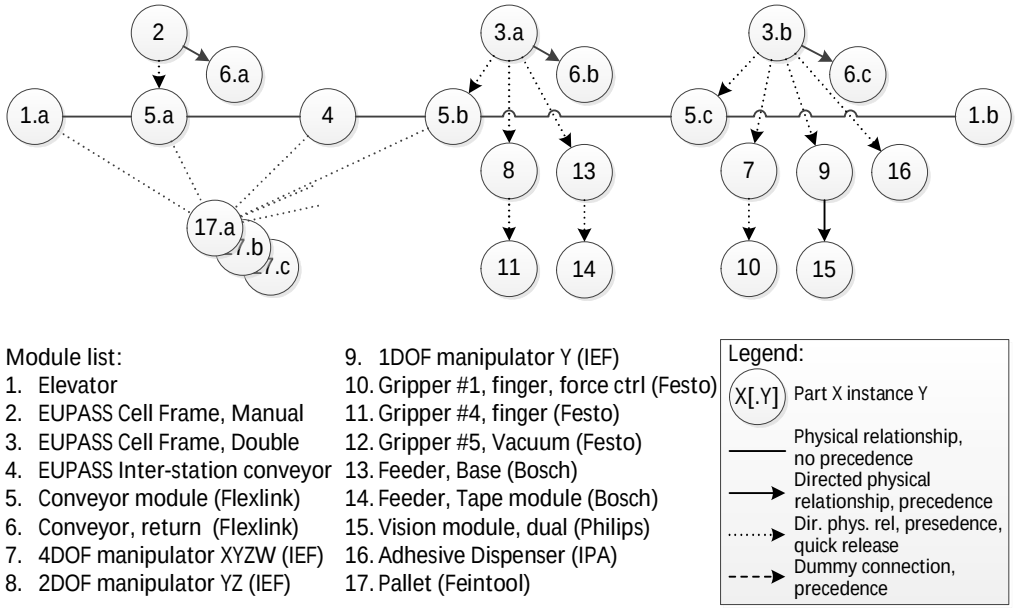


Figure 7.13: Production module liaison graph for CaseEnv 1 / valve

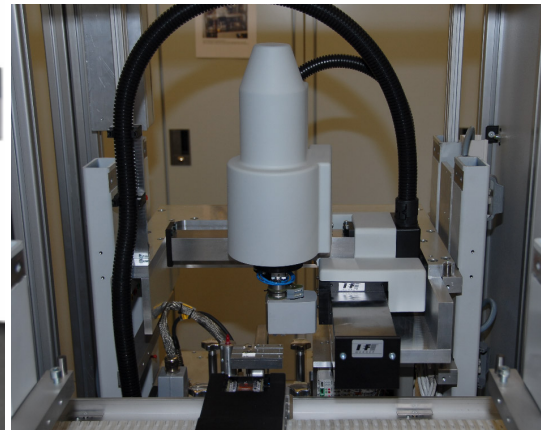
descriptions for these modules are defined through the Emplacement Concept, and BPs are created for each module type.

A few cells and modules from CaseEnv 1 are represented in Figure 7.14 (p.148). Figure 7.14a illustrates the overall view of the production system having the manual station (2) front right, followed by the two automatic cells (3.a) and (3.b) back left. Figure 7.14b represents the interior of the dispensing and assembly cell (3.b). It shows the manipulator (7) with gripper (10) at the top-back, vision system (15) at the front, conveyor system module (5.c) at the front, and dispensing module bottom-back. Figures 7.14c to e demonstrate the parts of the concept from a single module's point of view. The virtual view of the feeder module is shown in Figure 7.14c. A digital representation as an excerpt of the BP file in Figure 7.14e, and lastly the real feeder module mounted into the assembly cell in Figure 7.14d.

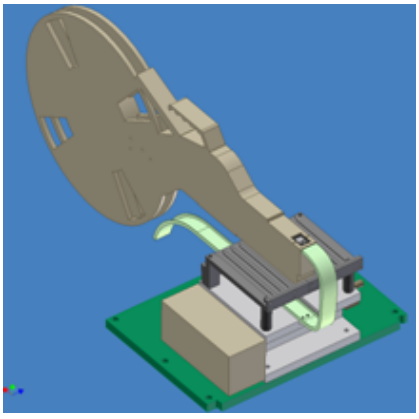
It is important to notice the architecture of the production cell, and the 'Bay Interface' concept [24] utilised within it. These define a rack-type tower structure, which has standardised positions (bay interface) on both sides of the cell - at front and back. A process module implementing the bay interface can be slid in and locked to any of these bays. In practice, the design of the process module restricts the possible bay slots where the module can be mounted, e.g. by the reachability of the assembled product. The material flow for the assembled product goes through the cell in the centre. Figures 7.15 and 7.16 illustrate the workstation framework and bay concepts. The implementation of the bay interface is visible in Figure 7.14b and Figure 7.14d. [27, pp.8-14] [30, pp.8-15] [23, 24]

For example, the cell (3.a) offers three bay interface positions per side for plugging in the process modules. The module's conveyor (5.b), manipulator (8) and feeder (13), all use the bay interface [24] for mechanical connection to the cell. This interface can be utilised to reconfigure the production cell quickly and easily for the next physical layout configuration and other system capabilities. The defined and standardised interface enables the digital description to state explicitly that it has a certain interface available in a specific pose. This information can be later utilised for mating the modules together into the reconfigured layout. Thus, both the cell BP and

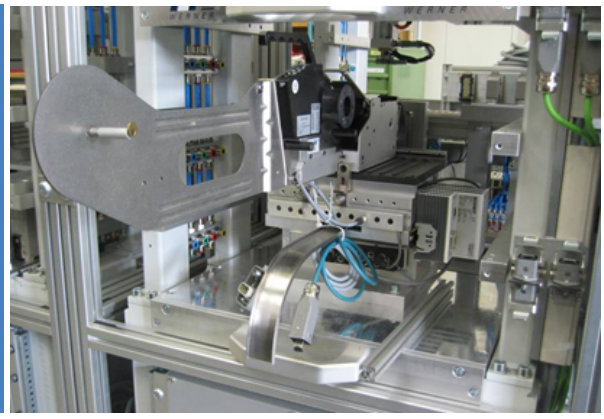
(a) EPS line for CaseEnv 1



(b) Dispensing cell including robot, dispensing, vision, and gripper modules



(c) CAD model of the feeder module



(d) Real feeder module mounted into cell

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="http://emplacementws.tte.tut.fi/EmplacementWS/stylesheets/BP_FormatterBPDoc.xsl"?>
<ModuleDescription xmlns:me="http://www.eupass.org/2005/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ww
    <Module category="PROCESS_MODULE" id="Bosch_PV2_TapeFeeder_Unit_APA_CX3467">
        <Info>
            <Label>
            <Comment>
                <String lang="en">Tape Feeder Unit for EUPASS PV2: Tape: 24mm. - on external request: perform tape feed by one mould - se
            </Comment>
            <Documentation>
            <PhysicalProperties>
                <Mass unit="kg" value="15"/>
                <InstallationPositions limited="false"/>
                <LocalOrigin iFInstanceRef="IF1"/>
                <BoundingBox decade="0" unit="m" increment="0.001">
                    <Min x="-0.450" y="-0.015" z="-0.010"/>
                    <Max x="0.100" y="0.015" z="0.300"/>
                </BoundingBox>
                <CentreOfGravity x="250.0" y="150.0" z="110.0"/>
            </PhysicalProperties>
            <EnvironmentalProperties>
                <EnvironmentalProperty classOfProperty="NORMAL_OP" datatype="LREAL" description="Acceptable humidity range for the module
                <EnvironmentalProperty classOfProperty="NORMAL_OP" datatype="LREAL" description="Acceptable shock range for the module du
                <EnvironmentalProperty classOfProperty="NORMAL_OP" datatype="LREAL" description="Acceptable temperature range during the
```

(e) BP excerpt of the feeder module i.e. it's digital representation

Figure 7.14: Sample cells and modules from CaseEnv 1

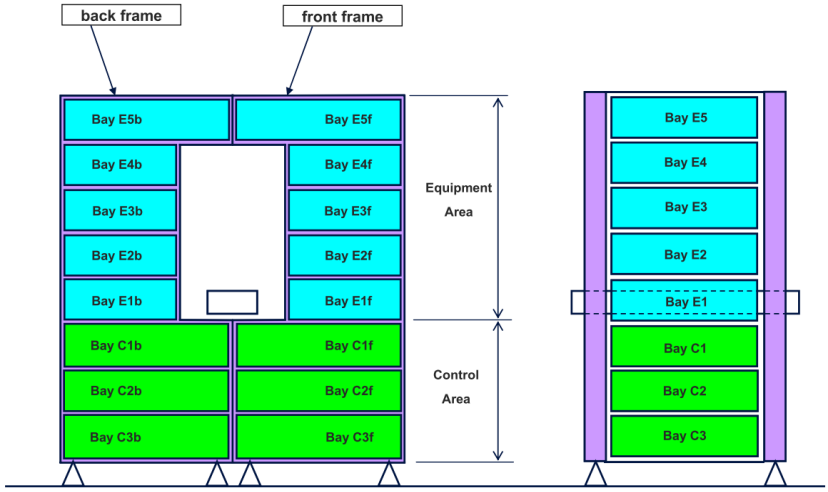
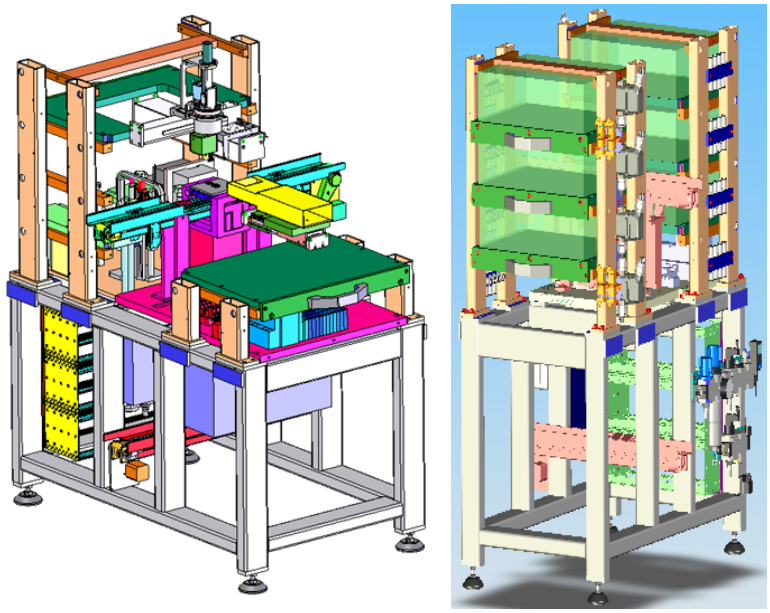


Figure 7.15: Illustration of cell architecture and positions for bay interface slots [27, Fig.1.3.3 p.10]



(a) CaseEnv 1 / assembly cell 3.b - Overview of interiors

(b) Example of double sided automatic cell frame with six process bay plates installed (green) and two pallet conveyors in the centre (red)

Figure 7.16: Examples of CaseEnv 1 cell architecture, framework, and bay concept

these three module BPs can all claim use of the same interface, although with different genders. In a similar manner, there is a shared interface specification between: a) the conveyor modules (1, 4 and 5) and the pallets (17) by utilising interface [22], and b) manipulator modules (7 and 8) and grippers (10 to 12) by utilising interface [61].

The complex capabilities can be produced from the BP information combined with the physical layout information, in a similar way that the interfaces are joined. The modules, especially the BP files in this case, provide simple capabilities, which then can be combined and deduced, with the help of the physical relations between the modules, into combined capabilities. These capabilities then define the temporal capability of the entire production system. However, combining capabilities is not as straightforward a task as stating that two interfaces are connectable. Additional rules and reasoning are needed to produce the resulting complex capabilities [65].

Digital Representation of Resources

In total, 17 different kinds of physical HW modules (24 individual units) were developed, manufactured and utilised in CaseEnv 1. Thus, 17 separate BP files were developed, each of which represents a different kind of production module. The key information is illustrated in Table 7.3, with a selection of these 17 Blueprints.

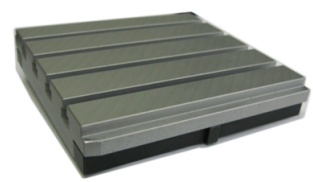
Table 7.3 and Table 8.1 (below) are composed from sections separated with page width lines. Each section represents a BP description with its key identification data. The first row starts with the running-order number and the *name* of the BP. The next rows start with a label and associated data: a) *BP ID* is the identification key of the BP description; b) *Category* is category and granularity level of the module; c) *Empl. ID* and *Empl Prof. ID* are back references to the Emplacement and $\text{Profile}_{\text{Empl}}$ which this BP is implementing; d) *Vendor* is the name of the Module Provider; e) *BP File* gives the BP file name and the URL link to the actual BP description file, with its full content; f) *Description* provides a short description of the module and its use, and g) a picture of the module is presented on the right. (BP file can contain links to external resources such as images and CAD files). All of the information listed above is directly retrieved from inside the BP description. The two additional fields are conducted externally for this thesis, for making visible the magnitudes for evaluation of the proposed concept: a) *Size* reports the BP file size in kilobytes and in lines, and b) *Instances* reports how many individuals of this kind of module existed in a corresponding case environment.

In order to make these BP files, the Emplacement level descriptions needed to be developed in advance or in parallel. Thus, ten Emplacements containing a total of 28 $\text{Profile}_{\text{Empl}}$s were developed (See Table 7.4). There were 11 extra $\text{Profile}_{\text{Empl}}$s developed as variants of the used ones, and available for future use.

Table 7.3: A selection of Blueprints developed for CaseEnv 1

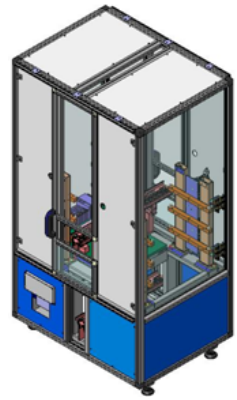
1. Feintool Carrier for EUPASS PV2 Size100

BP ID: Feintool_Carrier_for_EUPASS_PV2_Size100
Category: OTHER
Empl. ID: carrier.1
Empl Prof.ID: prof.EUPASS.Carrier.1
Size: 19,8 KB / 221 lines
Instances: 5 pcs
Vendor: Feintool
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_Feintool_PV2_Carrier.xml`



Description: Feintool Carrier for EUPASS PV2. Size 100.

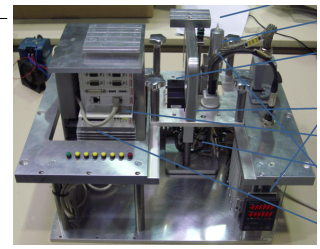
3. Feintool WSFrame SAAS 2-BayTower for EUPASS PV2



BP ID: Feintool_WSFrame_SAAS_2-BayTower_for_EUPASS_PV2
Category: PROCESS MODULE
Empl. ID: cell.1
Empl Prof.ID: prof.EUPASS_Cell.1
Size: 42,1 KB / 488 lines
Instances: 2 pcs
Vendor: Feintool
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_Feintool_PV2_Frame_SAAS_2-tower.xml`

Description: Feintool Workstation Frame for EUPASS PV2. Single Automatic Assembly Station (SAAS). 2-BayTowers. Local origin is located on the front left corner of the frame top.

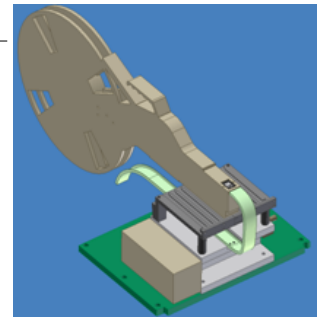
5. IPA Adhesive Dispenser for EUPASS PV2



BP ID: IPA_Adhesive_Dispenser_for_EUPASS_PV2
Category: PROCESS MODULE
Empl. ID: dispenser.1
Empl Prof.ID: prof.dispenser.needle.posCompensated.1
Size: 44,5 KB / 725 lines
Instances: 1 pcs
Vendor: Fraunhofer_IPA
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_IPA_PV2_Dispenser.xml`

Description: IPA Adhesive Dispenser for EUPASS PV2. Upside down needle adhesive dispenser with position compensation. Has laser based position measurement sensor. / Module is first scanning the surface of the workpiece and afterwards applying a continous stripe of glue following a specified dispense path on the workpiece. Hereby, the distance needle tip and workpiece is maintained constant.

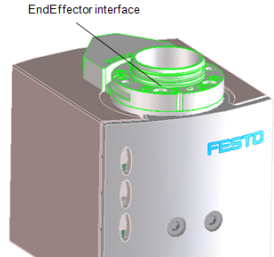
6. Bosch PV2 TapeFeeder Unit APA CX3467



BP ID: Bosch_PV2_TapeFeeder_Unit_APA_CX3467
Category: PROCESS MODULE
Empl. ID: feeder.1
Empl Prof.ID: prof.EUPASS.unit.feeder.tape.1
Size: 37,0 KB / 549 lines
Instances: 1 pcs
Vendor: RobertBosch
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_Bosch_PV2_TapeFeeder.xml`

Description: Tape Feeder Unit for EUPASS PV2: Tape: 24mm. - on external request: perform tape feed by one mould - send ready-signal when feeding part is in pick position

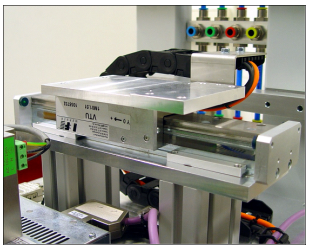
7. Festo Gripper no1 for EUPASS PV2



BP ID: Festo_Gripper_no1_for_EUPASS_PV2
Category: PROCESS MODULE

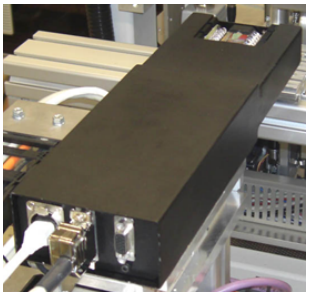
Empl. ID: gripper.1
Empl Prof.ID: prof.gripper.2-finger_forceCtrl.1
Size: 45,7 KB / 785 lines
Instances: 1 pcs
Vendor: Festo_AG_Co._KG
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_Festo_PV2_Gripper1.xml`
Description: Festo Gripper No1. for EUPASS PV2. - Two finger tactile gripper with force control.

10. IEF manipulator 1DOF for EUPASS PV2



BP ID: IEF_manipulator_1DOF_for_EUPASS_PV2
Category: PROCESS MODULE
Empl. ID: manipulator.1
Empl Prof.ID: prof.robot.cartesian.1DOF.X.1
Size: 39,9 KB / 747 lines
Instances: 1 pcs
Vendor: IEF
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_IEF_PV2_Vision_Transportation_System.xml`
Description: IEF manipulator for EUPASS PV2. 1DOF. Type: euroLINE 32 KL. Vision Transportation Unit (VTU)

17. Philips Vision System 2-view for EUPASS PV2



BP ID: Philips_Vision_System_2-view_for_EUPASS_PV2
Category: PROCESS MODULE
Empl. ID: vision.1
Empl Prof.ID: prof.vision.EUPASS.camera.two-view.1
Size: 29,1 KB / 406 lines
Instances: 1 pcs
Vendor: Philips
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_Philips_PV2_Vision_System_2-view.xml`
Description: Philips Vision System for EUPASS PV2. 2-views. Alignment purposes. The origin of the module is located on the center of the attachment interface.

7.5 Developing the Emplacement Descriptions

The iterative development cycles were followed in the CaseEnv 1. The data models evolved from bottom-up iterations over time. Initial data models which met the prevailing requirements were defined for both the BP and Emplacement. Data about the production modules was gathered and fed into the BPs and then generalised to the Emplacements. The same iteration loop was repeated for three major iterations: PV1, the mid-term workshop, and PV2. This approach has been applied for the core and most representative modules of the environment. In the finishing

phase of CaseEnv 1, the direction of the loop was changed to top-down. The last production modules were first modelled at the general level Emplacement, from which the customised BP Skeleton was automatically generated, and finally the module details were placed in the BP description document.

In parallel to this development, a questionnaire of applicable standards was launched for the project consortia and the general public. The intention was to capture mainly interface, communication, and architecture-related standards applicable for the Emplacement Concept, the implementation of CaseEnv 1, and modular production systems in general. In the end, the information also included the standards related to the safety, production processes and mechanical components. Altogether, 213 standards were listed, half of which were analysed and evaluated in detail. This information was utilised when developing and creating the Emplacement and the BP instance documents. Standards judged to be highly valuable were utilised and referenced as interfaces for mechanical, electrical, and communication connections; information exchange protocols, and mechanical components used in the implementation of both interfaces and modules.

In the verification case, CaseEnv 2, a top-down approach was followed. The generalisations, i.e. Emplacements, were first modified to model the general descriptions and definitions of the production modules. The BP Skeletons were generated, and subsequently filled with the details corresponding to the digital representation of the production module. This represents the standard practice and intended process flow for making the production module descriptions in the future.

Table 7.4 shows the Emplacements developed in the two use-cases presented here. The table represents, in each line separated section, the content summary of a single Emplacement. The first row starts with the ID of the Emplacement. Next, the P_I reports the number of implementable $\text{Profile}_{\text{Empl}}$s, followed by ($P_{All}$) reporting the sum of all $\text{Profile}_{\text{Empl}}$s defined in the Emplacement. Finally, the level defines what level of granularity the Emplacement focuses on. The followed $\text{Profile}_{\text{Empl}}$ data rows starts with the symbol '$\llcorner$', in order to separate the two different kinds of data rows presented in the table. The latter starts with the $\text{Profile}_{\text{Empl}}$ ID, followed by a character indicating whether this profile is an implementable one (I) or an abstract one (A). The next number, labelled as case, indicates which of the use-case environments this $\text{Profile}_{\text{Empl}}$ is mainly intended for. 1 means CaseEnv 1, 2 is CaseEnv 2, and 0 means the $\text{Profile}_{\text{Empl}}$ is used in both case environments or is in reserve. Finally, the row ends with the description of the $\text{Profile}_{\text{Empl}}$.

The digital representation for CaseEnv 1 included ten Emplacements containing a total of 28 implementable $\text{Profile}_{\text{Empl}}$s (See Table 7.4). The table excludes the abstract $\text{Profile}_{\text{Empl}}$s. Altogether, after both case studies, eleven Emplacements were developed containing a total of 132 different $\text{Profile}_{\text{Empl}}$s, of which 35 are implementable ones. Table 8.4 (p.164) reports the final sizes and number of rows for each Emplacement description.

Table 7.4: Emplacement and Profiles developed in Use Cases

Emplacement ID $\llcorner$ **Profile ID**	$P_I(P_{All})$ **A/I**	**Case**	**Level** **Description**
axis.1		1(5)	DEVICE
$\llcorner$ prof.cartesian_axis.1	I	0	The profile for Cartesian 1DOF axis
carrier.1		1(2)	ELEMENT
$\llcorner$ prof.EUPASS.Carrier.1	I	1	Specification for the EUPASS carrier
cell.1		2(7)	CELL
$\llcorner$ prof.EUPASS_Cell.1	I	1	Specification for the EUPASS assembly cell framework

Continued on next page

Table 7.4 – *Continued from previous page*

Emplacement ID └ **Profile ID**	$P_I(P_{All})$ **A/I**	**Case**	**Level** **Description**
└ prof.TUTuF_Cell.1	I	2	Specification for the TUT microfactory cell framework
controller.1		2(8)	UNIT
└ prof.EUPASS_Controller.1	I	1	Specification for the EUPASS controller module
└ prof.TUTuF_Controller.1	I	2	Specification for the TUT microfactory controller module
dispenser.1		2(8)	UNIT
└ prof.dispenser.needle.1	I	1	Profile for needle dispenser.
└ prof.dispenser.needle.posCompensated.1	I	1	Profile for position compensated needle dispenser.
feeder.1		5(15)	DEVICE
└ prof.EUPASS.module.feeder.tape.1	I	1	The profile for module level tape feeder for EUPASS PV2
└ prof.EUPASS.module.feederbank.1	I	1	The profile for module level feeder bank for EUPASS PV2
└ prof.EUPASS.unit.feeder.tape.1	I	1	The profile for unit level tape feeder for EUPASS PV2
└ prof.feeder.tape.1	I	0	The profile for tape feeder
└ prof.tape.EIA_418_B.1	I	0	Definition of the Tape and Reel used to deliver components to tape feeders.
gripper.1		5(14)	DEVICE
└ prof.gripper.2-finger_forceCtrl.1	I	0	This is the profile defining the required interfaces and properties for force controlled 2-finger grippers.
└ prof.gripper.2-finger_positionCtrl.1	I	0	This is the profile defining the required interfaces and properties for position controlled 2 finger grippers.
└ prof.gripper.2-finger_simple.1	I	0	This is the profile defining the required interfaces and properties for simple 2 finger grippers.
└ prof.gripper.3-finger_simple.1	I	0	This is the profile defining the required interfaces and properties for simple 3 finger grippers. Angle between fingers are 120$degree$. Every finger is moving simultaneously.
└ prof.gripper.vacuum.1	I	0	This is the profile defining the required interfaces and properties for vacuum grippers.
manipulator.1		8(26)	UNIT
└ prof.robot.PKM.3DOF.XYZ.1	I	0	Profile for Parallel Kinematics Mechanics (PKM) manipulator
└ prof.robot.cartesian.1DOF.X.1	I	0	Profile for cartesian manipulator
└ prof.robot.cartesian.2DOF.XY.1	I	0	Profile for cartesian manipulator
└ prof.robot.cartesian.3DOF.XYZ.1	I	1	Profile for cartesian manipulator. 3DOF. XYZ axes. EUPASS Bay interface.

Continued on next page

Table 7.4 – *Continued from previous page*

Emplacement ID ∟ **Profile ID**	$P_I(P_{All})$ **A/I**	**Case**	**Level** **Description**
∟ prof.robot.cartesian.3DOF.XYZ.2	I	2	Profile for cartesian manipulator. 3DOF. XYZ axes. TUT uF process interface(s)
∟ prof.robot.cartesian.4DOF.XYZW.1	I	1	Profile for cartesian manipulator. 4DOF. XYZW axes. EUPASS Bay interface.
∟ prof.robot.cartesian.4DOF.XYZW.2	I	2	Profile for cartesian manipulator. 4DOF. XYZW axes. TUT uF process interface(s)
∟ prof.robot.scara.3DOF.J1J2Z.1	I	0	Profile for SCARA manipulator
pickAndPlace.1		1(3)	UNIT
∟ prof.robot.rectangular.1	I	0	Profile for rectangular(cartesian) manipulator
transporter.1		3(17)	UNIT
∟ prof.transporter.EUPASS.interCell.1	I	1	The profile for FIFO type EUPASS Transporter connecting two cells together.
∟ prof.transporter.EUPASS.intraCell.mainLane.1	I	1	The profile for FIFO type EUPASS Transporter for intra cell operations on the main process flow
∟ prof.transporter.EUPASS.intraCell.returnLane.1	I	1	The profile for FIFO type EUPASS Transporter for intra cell operations on the return process flow
vision.1		5(27)	DEVICE
∟ prof.vision.EUPASS.camera.two-view.1	I	1	EUPASS - Two directional view camera for e.g. aligning processes
∟ prof.vision.camera.1	I	0	One directional view camera for vision operations. Used e.g for inspection, measurement. Integrated optics.
∟ prof.vision.camera.2	I	0	One directional view camera (without optics) for vision operations. Used e.g for inspection, measurement. Another interface used for mounting lenses.
∟ prof.vision.camera.two-view. 1	I	0	Two directional view camera for e.g. aligning processes
∟ prof.vision.lens.1	I	0	Lens with C mount
$\sum_{Empl}$ = **11**		$\sum_{Profile}$ = **35(132)**	

8 Verification and Validation

Verification, in the context of this thesis, means a process for evaluating whether or not a product, service or system complies with a regulation, requirement, specification, or imposed condition. It is an internal process. Similarly, validation is the process for assuring that a product, service, or system meets the needs of the end user and other identified stakeholders. It involves acceptance and suitability with external customers. On the whole, verification is the main focus here.

The second case study is the TUT μFactory®, which is used for verification and partial validation of the proposed concept. TUT μFactory has been an R&D environment for ongoing research around desktop and microfactory concepts since 2005 at TUT / Department of Production Engineering (TTE), and has continued at TUT / Department of Mechanical Engineering and Industrial Systems (MEI). In this chapter, the verification criteria is first defined, followed by an introduction to the TUT μFactory environment. Next, the concept is used to develop module descriptions for modules in the TUT μFactory environment. Finally, the validation results are presented and analysed.

8.1 Verification Criteria

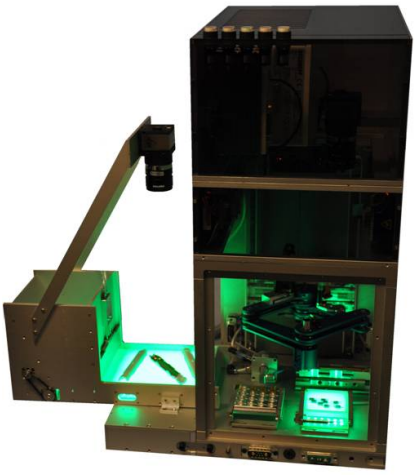
The objective is to prove that the concept works without any need for modifications, and is capable of expressing different modules in completely different architectures and domains. The concept will only be verified if, and only if, the verification shows or satisfies that:

1. The modules of the verification environment can be modelled with the proposed concept, and with the given set of model definitions and file formats.
2. The description data models (i.e. XSDs) are not changed at all, or at least require no major changes. This means, in practice, that the semantics of the description file formats hold, and they are capable of expressing different kinds of modules from different kinds of implementation architectures.
3. The supporting and processing related files (e.g. XSLTs / See Ch. 7.2 (p.136)) should not be changed. However, in this case changes are more acceptable, as these are related to, for example, output, layout, and the structure of the documentation. If the verification requirement §2 needs any change, those changes will be reflected here.
4. The Emplacement files may be changed by adding new $\text{Profile}_{\text{Empl}}$s as the verification environment is based on completely different architectures and interfaces. However, the amount of changes in total is expected to be low. Previously created $\text{Profile}_{\text{Empl}}$s should remain unaffected.
5. No, or at worst only a few, completely new Emplacement files need to be created. This will happen, though, if completely new processes appear out of the set of processes defined by the development CaseEnv 1.
6. A completely new set of BP descriptions are generated for expressing new modules in the verification environment (CaseEnv 2).

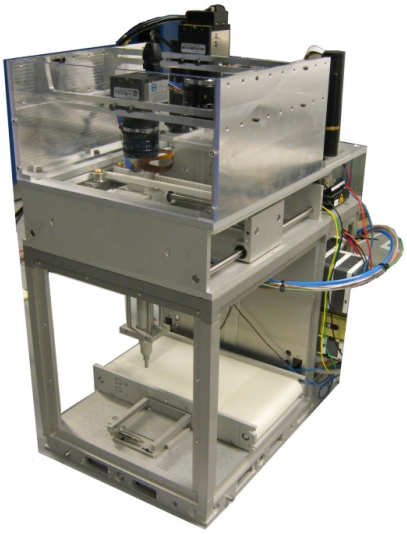
8.2 Case 2: Verification / TUT microFactory Environment

The TUT μFactory environment (CaseEnv 2) is used as a verification and validation platform for the developed Emplacement Concept. As the concept is developed against CaseEnv 1 (Ch. 7.4), this functions as verification for the case study (CaseEnv 2). The TUT μFactory concept was first introduced in 2005, and research around micro- and desktop-factories at TUT dates back to 1999 [44, 177]. It is a modular concept for desktop-sized production systems. The architecture and interface definitions of TUT μFactory are discussed in [120].

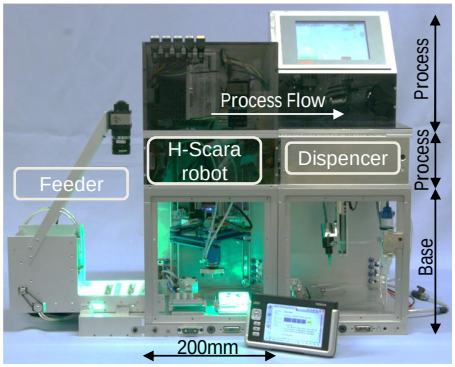
CaseEnv 2 focuses on a TUT μFactory production line used to assemble a gas sensor. It consists of two cells, which are built from a total of eight modules (Figure 8.1d). The modules include two base units, two manipulators, one feeder, one gripper and two combined controller/HMI/vision units. The overall dimensions of the production line are 610×300×500 mm (L×D×H), and one man can move it easily in its entirety. [122]



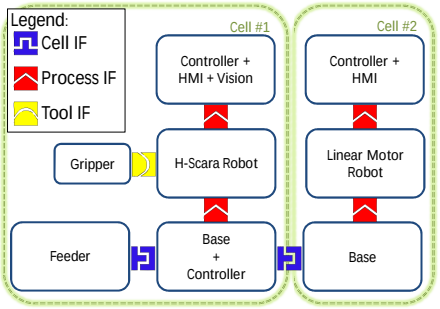
(a) TUT μFactory: H-Scara cell with feeder



(b) TUT μFactory: Screw insertion cell



(c) TUT μFactory: Sensor assembly line [122]



(d) Sensor assembly line: the modules and the interfaces [122]

Figure 8.1: Sample cells and modules from the CaseEnv 2

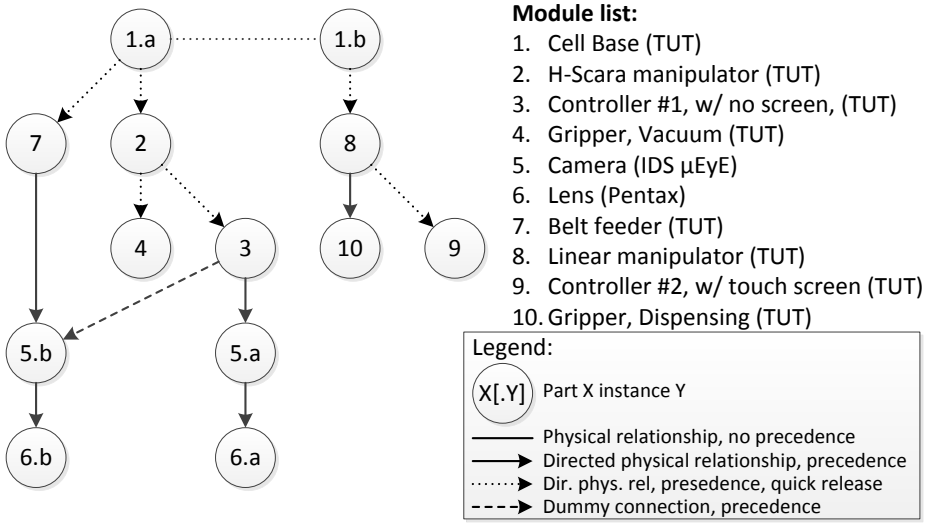


Figure 8.2: CaseEnv 2 liaison graph

A few figures, illustrating lines, cells and modules from the TUT μFactory projects, are represented in Figure 8.1, of which a), c) and d) are directly related to the module descriptions made for CaseEnv 2. Figure 8.1c represents the modular sensor assembly line, which is CaseEnv 2. Figure 8.1a illustrates the first assembly cell (the tower on the right) and the belt feeder attached into it (left). The tower is assembled as a sandwich structure by repeating the TUT μFactory process interfaces [133] on top of each other. The modules and their interfaces are demonstrated in Figure 8.1d. Figure 8.1b shows a modular, desktop screw-insertion cell following the TUT μFactory architecture.

8.2.1 System Components and Layout

Figure 8.2 represents the modules and the liaison graph for CaseEnv 2, the system illustrated in Figure 8.1c. The system is composed of two assembly cells and a belt feeder. The first cell is composed of base frame (1.a), on top of which is mounted a 4DOF H-Scara manipulator (2) with a vacuum gripper (4). The manipulator uses a TUT μFactory process interface [133] to connect modules both underneath and above it, which creates the sandwich structure. On top of the manipulator there is a controller and HMI module (3), which has a camera (5.a) with a lens (6.a). The belt feeder (7) is connected to the right cell interface [132] of (1.a). The feeder holds another camera (5.b) with a lens (6.b). The camera (5.b) is wired to controller (3), i.e. it has an electrical and SW connection to it. The second cell is composed of the base frame (1.b), a 3DOF cartesian manipulator (8) attached on top of the base, and a dispensing head (10) mounted on the manipulator as a tool. Another controller module with touch screen HMI (9) is attached on top of the manipulator (8). The cells are connected together with the cell interface [132], i.e. the left cell interface of (1.a) is connected with the right cell interface of (1.b). The physical system layout can be quickly and freely modified at the defined interfaces [61, 132, 133] according to the architectural definitions [120]. Figure 8.1d highlights the modules and their interfaces.

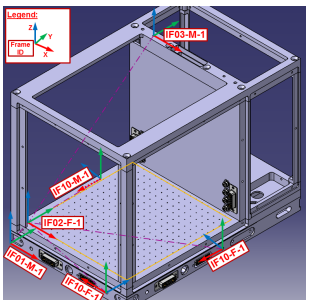
8.2.2 Digital Representation of Resources

A few changes to existing Emplacements were needed and new $\text{Profile}_{\text{Empl}}$s had to be developed and added for CaseEnv 2, because of the different architecture and interfaces compared to CaseEnv 1. The details and dimensions of these changes are discussed in Ch. 8.3. A few $\text{Profile}_{\text{Empl}}$s were directly applicable, such as gripper for instance, because it follows the same interface definition as used in CaseEnv 1, but just in a smaller size, meaning it is just another class of the same interface specification. For the sake of space, Table 7.4 above also shows the Emplacements and $\text{Profile}_{\text{Empl}}$s used in CaseEnv 2.

The complete sensor assembly system consists of 13 individual, physical HW units of 10 different kinds. A subset of these were selected for CaseEnv 2. Altogether, it includes and models eight different kind of physical HW modules (11 individual units). Thus, eight separate BP files were developed, each of which represents a different kind of module. The key information from these Blueprints is illustrated in Table 8.1. The belt feeder (7) and the dispensing head (10) were not modelled because of the lack of generalisation potential, and time. The table uses the same formatting as the earlier table, Table 7.3, described in Ch. 7.4 (p.150).

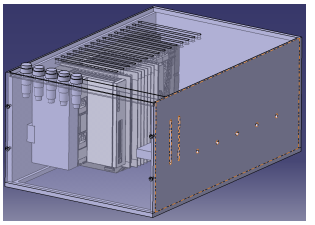
Table 8.1: A selection of Blueprints developed for CaseEnv 2

5. TUT uF BaseFrame



BP ID: TUT_uF_BaseFrame
Category: PROCESS MODULE
Empl. ID: cell.1
Empl Prof.ID: prof.TUTuF_Cell.1
Size: 41,6 KB / 453 lines
Instances: 2 pcs
Vendor: TUT
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_TUT_uF_BaseFrame.xml`
Description: TUT microfactory cell base frame. Creates the base foundation for a cell. Process is implemented inside the cell.

6. BP TUT uF Controller-1



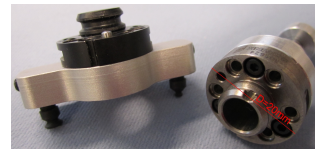
BP ID: BP_TUT_uF_Controller-1
Category: PROCESS MODULE
Empl. ID: controller.1
Empl Prof.ID: prof.TUTuF_Controller.1
Size: 33,1 KB / 393 lines
Instances: 1 pcs
Vendor: TUT
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_TUT_uF_Controller-1.xml`
Description: TUT microfactory controller No1. Basic controller hardware package.

7. BP TUT uF Controller-2

BP ID: BP_TUT_uF_Controller-2
Category: PROCESS MODULE
Empl. ID: controller.1

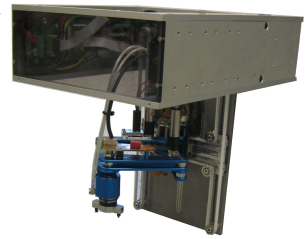
Empl Prof.ID: prof.TUTuF_Controller.1
Size: 32,6 KB / 384 lines
Instances: 1 pcs
Vendor: TUT
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_TUT_uF_Controller-2.xml`
Description: TUT microfactory controller No2. Basic controller hardware package.

13. TUT uF Gripper Vacuum 1



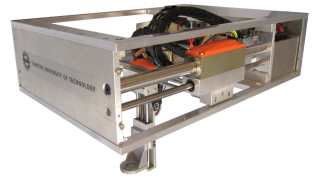
BP ID: TUT_uF_Gripper_Vacuum_1
Category: SUB MODULE
Empl. ID: gripper.1
Empl Prof.ID: prof.gripper.vacuum.1
Size: 41,4 KB / 587 lines
Instances: 1 pcs
Vendor: TUT
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_TUT_Gripper_Vacuum_1.xml`
Description: TUT microfactory Gripper - Vacuum - No1. Mounting: ISO 29262. Two suction cups at bottom side.

17. TUT uF Manipulator H-Scara



BP ID: TUT_uF_Manipulator_H-Scara
Category: PROCESS MODULE
Empl. ID: manipulator.1
Empl Prof.ID: prof.robot.cartesian.4DOF.XYZW.2
Size: 53,0 KB / 832 lines
Instances: 1 pcs
Vendor: TUT
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_TUT_Manipulator_H-Scara.xml`
Description: TUT microfactory H-Scara manipulator module. Double parallel kinematics manipulator module. 4DOF. XYZW. It has two parallel kinematics structures mounted in series. First H-structure providing XZ movement and on it is mounted parallel scara structure providing XY movement.

18. TUT uF Manipulator Linear



BP ID: TUT_uF_Manipulator_Linear
Category: PROCESS MODULE
Empl. ID: manipulator.1
Empl Prof.ID: prof.robot.cartesian.3DOF.XYZ.2
Size: 55,3 KB / 869 lines
Instances: 1 pcs
Vendor: TUT
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_TUT_Manipulator_Linear.xml`

Description: TUT microfactory Linear manipulator module. Cartesian gantry type of manipulator module. 3DOF. XYZ. Kinematics: Linear motor Y, Linearmotor X and linear motion (ball screw) Z.

23. IDS UI-1540SE-M-GL

BP ID: IDS_UI-1540SE-M-GL
Category: SUB MODULE
Empl. ID: vision.1
Empl Prof.ID: prof.vision.camera.2
Size: 49,4 KB / 690 lines
Instances: 2 pcs
Vendor: IDS_Imaging_Development_Systems_GmbH
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_camera_IDS_UI-1540SE-M-GL.xml`
Description: IDS camera. model:UI-1540SE-M-GL

24. Pentax TV-lens H1214-M

BP ID: Pentax_TV-lens_H1214-M
Category: SUB MODULE
Empl. ID: vision.1
Empl Prof.ID: prof.vision.lens.1
Size: 33,5 KB / 372 lines
Instances: 2 pcs
Vendor: Pentax
BP File: `http://emplacementws.tte.tut.fi/EmplacementWS/bluePrints/BP_lens_Pentax_TV-lens_H1214-M.xml`
Description: Pentax lens. model:TV lens / H1214-M

8.3 Analysis of Results from the Verification

It was possible to describe all the modules from both case environments with the Emplacement Concept. This consists of a total of 25 different BPs containing 35 module instances. The detailed figures for each case environment are given in Table 8.2. These BPs have been governed by 11 Emplacement descriptions containing 35 implementable $\text{Profile}_{\text{Empl}}$s and 132 abstract $\text{Profile}_{\text{Empl}}$s (See Table 7.4 (p.153)). This demonstrates the capabilities of the Emplacement Concept, associated information models and description formats (Emplacement and BP), and the developed tools.

Table 8.2: Number of BPs and module instances in case environments

	BPs	Module instances
CaseEnv 1	17	24
CaseEnv 2	8	11
Total	25	35

Table 8.3 and Table 8.4 compare the statuses of both data model implementations and Emplacement descriptions after the two case studies. These tables summarise the changes made to

Table 8.3: Changes made to the XSDs and the XSLTs files

Filename	Final		Difference		Diff. %	
	Bytes	Lines	Bytes	Lines	Bytes	Lines
HistoryContainer_v1-0-0.xsd	10980	271	10980	271	NA %	NA %
EUPASSStandard.xsd	3009	77	1302	39	76,27 %	102,63 %
EUPASSBluePrint_v1-0-0.xsd	37780	873	2703	57	7,71 %	6,99 %
EUPASSBaseTypes_v1-0-0.xsd	46695	1123	3027	63	6,93 %	5,94 %
EUPASSEmplacement_v1-0-0.xsd	16808	401	0	0	0 %	0 %
EUPASSRepository.xsd	14109	341	0	0	0 %	0 %
EUPASSSkills_v1-1-0.xsd	4052	98	0	0	0 %	0 %
EUPASSAssemblySystem_v1-0-0.xsd	12326	290	0	0	0 %	0 %
BP_post-processing_v1-0-0.xsl	2582	62	2582	62	NA %	NA %
BP_FormatterBPDoc_v1-0-0.xsl	64744	2042	10465	270	19,28 %	15,24 %
Empl_FormatterBPSkeleton_v1-0-0.xsl	35398	814	5694	125	19,17 %	18,14 %
Empl_FormatterEmplDoc_v1-0-0.xsl	52705	1653	344	12	0,66 %	0,73 %

different files within the concept after moving from CaseEnv 1 to CaseEnv 2. The results are used to verify the generalisability of the proposed solution, according to the verification criteria specified in Ch. 8.1.

The situation after CaseEnv 1 is selected as the baseline (figures for this state are not shown in the tables). The status of the files after completion of CaseEnv 2 is presented as state *Final*. Figures are given in two units – the file size in bytes and the number of lines in the file. The *difference* shows the subtraction of the baseline from the final state (CaseEnv 2 - CaseEnv 1) in absolute values, and finally *Diff. %* reports the same difference compared to the baseline in percentages. *NA* means 'not available', in case the initial version after CaseEnv 1 is non-existent.

Table 8.3 compares the changes made to the data models (XSDs) of the various descriptions, and the transformations (XSLTs) used to alter the content of descriptions. The completion of the CaseEnv 1 descriptions fixes the baseline. The end point is after the CaseEnv 2 descriptions are completed and the required changes have been compared to the baseline. The Verification Criteria §2 and §3 (Ch. 8.1) are associated with this case, having the objective of no change, or only minor changes to take place.

The purpose and content of these files are as follows: *EUPASSBluePrint_v*.xsd* defines the information and data model for the BP descriptions. *EUPASSEmplacement_v*.xsd* models the Emplacement descriptions. *EUPASSBaseTypes_v*.xsd* contains shared and commonly used data models, much like a library of constructional components. This file is included by other definitions like *EUPASSBluePrint.xsd* and *EUPASSEmplacement.xsd*, which can then utilise the same models. This will increase the consistency of the definitions and facilitate maintenance. *EUPASSAssemblySystem_v*.xsd* defines the data model for the system layout, but it is not used in this thesis. The rest, *EUPASSStandard.xsd*, *EUPASSSkills_v*.xsd* and *EUPASSRepository.xsd*, are supporting data models. The first two are used as containers for storing interface standards and skill definitions, from where these specifications can be easily copied to the Emplacement files. The third is used for the internal implementation of EmplWS.

The following changes took place in the XSDs in Table 8.3. The two main elements changed in the *EUPASSBluePrint_v*.xsd* were – **`PhysicalProperties`** and **`Interface`**. In the case of **`Physical Properties`**, **`AccelerationLimits`** internal definition was changed and moved to *EUPASSBaseTypes.xsd* file. Furthermore **`LocalOrigin`** and **`BoundingBox`** were added as children of **`PhysicalProperties`**. The **`ForceAndTorqueLimits`** was added to the **`Interface`**,

Table 8.4: Changes made to Emplacements files

Filename	Final		Difference		Diff. %	
	Bytes	Lines	Bytes	Lines	Bytes	Lines
Controller_Emplacement.xml	33451	354	33451	354	NA %	NA %
Vision_Emplacement.xml	58836	780	26126	367	79,87 %	88,86 %
Cell_Emplacement.xml	42874	413	9408	96	28,11 %	30,28 %
Manipulator_Emplacement.xml	52253	717	4855	111	10,24 %	18,32 %
GripperEmplacement.xml	52668	765	3519	78	7,16 %	11,35 %
Transporter_Emplacement.xml	57451	809	489	65	0,86 %	8,74 %
Feeder_Emplacement.xml	47186	610	328	32	0,70 %	5,54 %
Dispenser_Emplacement.xml	35681	494	235	39	0,66 %	8,57 %
Axis_Emplacement.xml	22283	301	140	28	0,63 %	10,26 %
Pick_and_Place_Emplacement.xml	15218	272	78	20	0,52 %	7,94 %
Carrier_Emplacement.xml	19756	192	11	6	0,06 %	3,23 %

which made it applicable to all the included interface instances at once. Each interface instance has the possibility of overriding these limits, if required. Another element added to **`Interface/Instance`** was **`MatrixLocations`** and its corresponding complex type "MatrixLocations_CT" with its internal structure. This structure describes interfaces where the base unit of the interface is repeated in a matrix pattern. For example, this can describe a Lego block, where a notch represents the base unit of the interface, which is repeated in an $m \times n$ matrix. The requirement for this kind of description arose from the TUT μFactory base module, which has a platform for mounting process modules on a matrix type interface of fastening holes.

A few 'complex type' definitions were added to *EUPASSBaseTypes_v*.xsd* – one for 6DOF data limits, containing minimum and maximum boundaries of 6DOF data; one for 3DOF cartesian data; and one for 3DOF cartesian data limits (min and max values). There were also a few minor changes in the form of added annotation and added enumeration options for the accepted file formats.

EUPASSStandard.xsd can be discounted from the analysis, because it is a supporting data model and file format for developers. It establishes a library storing interface standards descriptions, which can then be copied to or included in the Emplacements descriptions.

In the case of the XSLT files, the changes arise from the changes in the XSDs. The *Empl_Formatter-BPSkeleton_v*.xsl* file is used to create the BP Skeleton from the selected Emplacement and $\text{Profile}_{\text{Empl}}$. As the *EUPASSBluePrint_v*.xsd* and *EUPASSBaseTypes_v*.xsd* had a few changes, those need to be reflected here as well. The same applies to *BP_FormatterBPDoc _v*.xsl*, which is used to create human-readable HTML document out of a BP description.

The *Empl_FormatterEmplDoc_v*.xsl* has only a few minor changes originating from the changes in *EUPASSBaseTypes_v*.xsd*. *BP_post-processing_v*.xsl* is a new and small supporting processor for BPs files. Its task is to sort 'Variable Details' and 'Interfaces' into a consistent order. The mandatory ones first appear in the listing, and then these are sorted in ascending order according to the `name` and `id`.

Table 8.4 compares the changes made to the Emplacements descriptions. Again, the baseline was fixed after the completion of the CaseEnv 1 descriptions and the final point is after completion of the CaseEnv 2 descriptions, after which the results of the latter are compared with the baseline. The Verification Criteria §4 to §5 (Ch. 8.1) are associated with this evaluation case.

The following changes are represented in Table 8.4. *Controller_Emplacement.xml* did not exist as an independent Emplacement in Case Env 1. The Case Env 2 needed it, so it was created with the necessary $\text{Profile}_{\text{Empl}}$s and other definitions. In the case of *Vision_Emplacement.xml*,

the changes originated from a thorough analysis of the cameras and vision systems with a domain expert (Thank you Timo), and by capturing this knowledge in the Emplacement description. This Emplacement was more or less a stub after Case 1, and therefore needed an update. The changes add ten new interfaces, standards and specifications, adding and modifying both properties and data types, adding 13 new $\text{Profile}_{\text{Empl}}$s, changing properties associated with the existing $\text{Profile}_{\text{Empl}}$s, and adding four new capabilities. Three interface standards and two new $\text{Profile}_{\text{Empl}}$s were added to *Cell_Emplacement.xml*, which represent the *TUT µFactory base frame*, *prof.TUTuF.genericBase.1* and *prof.TUTuF_Cell.1*. A few typos were corrected. In the case of *Manipulator_Emplacement.xml* the changes come from adding an interface standard related to the TUT µFactory, removing one obsolete interface, adding three $\text{Profile}_{\text{Empl}}$s, and appending a set of additional capabilities. In the case of *GripperEmplacement.xml* the changes come from adding a few terms, adding three interface standards, changing two interface standards to use classes and options, and updating $\text{Profile}_{\text{Empl}}$ *prof.gripper.vacuum.1* to include more optional interfaces.

In the cases from *Transporter_Emplacement.xml* to *Carrier_Emplacement.xml* there are only small changes. These include small structural and naming changes in the data model for Emplacements, *EUPASSEmplacement_v1-0-0.xsd*. These minor changes apply to all listed Emplacement descriptions.

9 Discussion

9.1 Evaluation of the Proposed Concept

The evaluation is performed along the right branch of the V-model, starting from implementation at the bottom. Then the results are verified against the requirements, layer by layer, finally ending up with a comparison of the highest level requirements, namely the objectives and Research Questions (RQs).

9.1.1 Analysis of Emplacement and Module Descriptions

It was possible to model all of the production modules, from both case environments, with the Emplacement Concept and its implemented file formats – BP and Emplacement. The formalised data models successfully captured the capabilities, interfaces, and technical and business-related properties of the modules. The system design and reconfiguration could be performed based on the supplied information. The required changes to the underlying data models were minimal or non-existent when describing the second case environment, which represented a completely different kind of production environment from the first case. These two aspects – the modelling power of expression in all cases and the fact that there was no need for any change – demonstrate the generalisability of the proposed Emplacement Concept and the implemented file formats.

The captured information is available in a standardised format for any system design or deployment tool, improving the exchange of information. The EmplWS acts as an on-line information-sharing service with both a human interface (web pages) and an application interface (WSDL). The information shared by the service is: a) data models (i.e. XSDs files); b) Emplacement files; c) BP files, and d) description file processing instructions (i.e. XSLTs). The human interface offers a few information search tools for looking for modules implementing specific capabilities or interfaces, and vice versa. These demonstrate how the Emplacement Concept facilitates the system design process. However, in the future, specific system design tools will provide more efficient methods for searching for and selecting fitting modules for the designed system. As analysed above, the Emplacement Concept and its implementations, Emplacement files, BP files, and associated tools, provide an answer to sub-objective 3.

The Status of Exchanged Documents The practical implementations of Emplacement and the BP formats are standardised by the EUPASS organisation [174] in [25, 26]. These are the implementations for the proposed conceptual formats of the Abstract Module Description and Module Description. A version of the History Container (22) has been designed, but the format is not yet standardised. Many of the exchange formats, represented in Figure 5.10 and Figure 5.11, are undefined, such as information and exchange models about the standards (12) and processes (11); Product-Process requirements (2); Layout (3), and Recipe (4) documents.

It is important to note, even though it is beyond the scope of this thesis, that currently many of the tools represented in the framework (Ch. 5.5) are non-existent. There are only a few prototypes representing partials of the tools, and the tools that are indispensable for the concept. However, there is not yet any complete and coherent path throughout the represented framework and system design and commission process. The situation will be improved with future work which will fill in the gaps of the missing tools, and will utilise the proposed formats.

9.1.2 Analysis and Evaluation of the Emplacement Concept and Generic Model

The system design framework is presented in Ch. 3.2 and Ch. 5.5. The Emplacement Concept, together with its Generic Model, provides a solution for the issues of production system design, and at the same time to sub-objective 2. An experimental evaluation of the benefits of the Emplacement Concept against the high level objectives is impossible at the moment. This is because the complete tool chain and identified process data exchange documents for the overall production system design framework (Ch. 5.5) are still missing. In addition, current implementations do not support the proposed Emplacement Concept and implemented module descriptions, yet.

Therefore, heuristic deduction is used to prove the objectives defined for the concept. Figure 4.3 in Ch. 4.4 showed a break-down of the goals of this work. Figure 9.1 ties the results of this thesis into the same map. These results are illustrated as green boxes at the bottom of the diagram. The results, Emplacement Concept and formal module descriptions (Emplacement and BP) are able to provide solutions for the goals in the bottom half of the map. It can thus be stated that the proposed concept fulfills these goals. As all the goals in the bottom layer have been achieved, the pointers and the following upstream goals have also been completed. There are paths from the proposed concept (the green rectangles) to the main goals of this thesis (the blue rectangles). This proves that the proposed concept fulfils the high level objectives defined at the beginning of the thesis.

9.1.3 Correspondence to the User Requirements and System Requirements

The identification of users and the definitions of URs and SRs (Ch. 4) correspond to sub-objective 1. These requirements were found out by analysing materials from workshops, conducting a thorough literature review and carrying out interviews with experts in the field. A full list of all the URs which were identified is given in Appendix B. The main URs of interest to this thesis are presented in Ch. 4.2.4.7.

The proposed Emplacement Concept and Generic Model, containing AMD, MD, and HC, fulfill and correspond to the main URs presented in Ch. 4.2.4.7. These definitions provide direct answers to UR1 to UR32 of the selected main URs. The Emplacement Concept together with the introduced tools meet UR60 and UR61.

The System Requirements are met as follows. The Emplacement Concept and proposed design framework (Ch. 5.5) provide answers to SR1 and SR2. The implementation of the Generic Model by Emplacement and BP file formats provides a common format acting between various systems, as requested by SR3. The BP and HC files are intended to travel physically with the production module, thus completing SR4. The Emplacement, BP, and HC file formats, in addition to selected implementations with XML technologies, provide solutions for SR5 to SR8. Formalism and decisions taken within the Emplacement Concept, Generic Model, and implementation of the data models by Emplacement, BP, and HC all together provide a solution for SR9 to SR13. However, accomplishing SR10 and SR11 needs some deliberate actions from the designer when describing the content, especially for the Emplacement files. This cannot be guaranteed a priori with the data models.

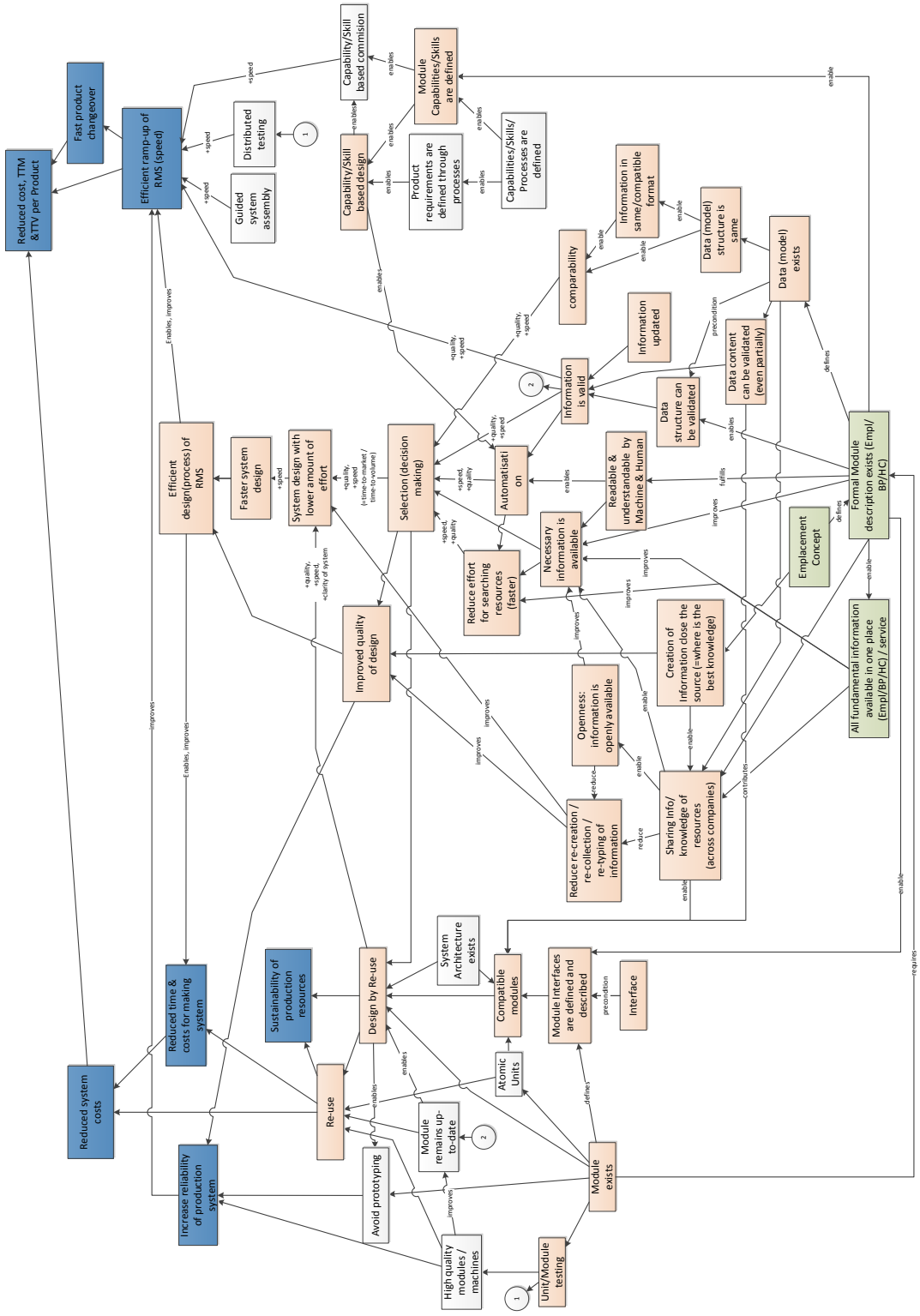


Figure 9.1: Goals and requirements breakdown map with results

9.1.4 Reflection on the Research Questions

The following answers can be found for the Research Questions presented in the beginning:

RQ0. *How to support and improve the design-by-reuse and module selection processes in designing and commissioning reconfigurable production systems?*
The Emplacement Concept supplies an answer and solution for this RQ. It proposes a concept that supports the system design process for RMS from already-available modules and thus promotes design-by-reuse at the same time.

RQ1. *How can a production system's resources be represented so that the needs of the different phases of system design can be met, along with the requirements of the identified users?*
The needs of modular system design were represented in the literature review. The different users and their roles were identified in Ch. 4.2.3, followed by identification of user requirements. From these, the design framework (Ch. 5.5) was produced, and the system design phases were identified on it. The Emplacement Concept is proposed in order to complete a few of the missing pieces in the framework, and to represent and model the production resources. It includes the Generic Model defined at the three abstraction levels – the AMD, MD, and MID.

RQ2. *What information about the production system modules needs to be collected and represented?*
The Generic Model complemented with the implementations of the Emplacement, the Blueprint, and the History Container file formats define the information model for production modules. These define the resource information in sufficient detail for designing and commissioning a production system for use. In addition to the comprehensive characterising of information about the production module, the importance of describing both capabilities and interfaces is highlighted.

RQ3. *How can the module descriptions be utilised by the different user roles in different phases of the production module's lifecycle?*
The utilisation section (Ch. 7) and the framework (Ch. 5.5) describe use case scenarios showing how the Emplacement Concept can be utilised in practice. The different tools which can be utilised are listed in Ch. 7.3, and Ch. 7.2 reports the ways to benefit from the descriptions in other environments.

9.2 Impact and Significance

The main impact of this thesis is its contribution to the formalization of information and the way it meets the application requirements of RMS and EPS. The formalised information describes production resources for systems which will become even more necessary in the future. Modern production systems have to respond better to challenges. As such, they need to be more productive, more agile, more responsive to change, provide shorter TTM and TTV, and at the same time they have to support human creativity. It is clear that the needs of the customers have to be reflected better in industrial production and this will only be possible with the next generation production equipment. This equipment is distinguished from current production systems in that we will know exactly what the modules are capable of, and we will be sure that they can be easily integrated into an operational production system, without too much effort or programming. In addition, the production modules can be re-used in the same manner as new ones, which increases their utility value as production assets and stops them depreciating in value so fast.

The full benefits of the proposed Emplacement Concept and resource data models can be harvested, when the concerned architectures and systems utilise modularity in all its aspects. This

means that the implemented production system is modular in its mechanical, electrical, service, communication and control systems. The mechanical modularity, including the mechanical connectors for other types of interfaces, is a primary condition for the applicability of the concept, and where to start with it. No modules can be integrated and interconnected into an operating system unless they have the same or compatible interface(s) and architecture(s). Nevertheless, the resource descriptions provided here can be utilised even if the above-mentioned conditions are not fulfilled, as the human system designer or integrator can benefit from the resource descriptions as semantic data sheets, which can be easily accessed and filtered. This will considerably improve the designers' efficiency, as less time is spent on searching for and retrieving information, and searching for, filtering, commensurating, comparing and selecting resources for the application.

The stakeholders who will benefit most from the proposed concept are large system integrators, system integrators with networked operating models, and large end-users. Large system integrators can internally develop their own system architecture, modularity, and interface concepts internally, although it is questionable whether they would reap that much benefit from formal descriptions of their own modules. There might be different methods for storing and maintaining the product-related information, for example in a Product Life Management (PLM) system. In any case, they would benefit from importing the formalised component descriptions from their suppliers.

A system integrator with a networked operation model, having a supplier network of SMEs or even larger companies could make better use of the proposed concept. In such a model there would be a system integrator utilising a variety of process modules from different origins. In such cases, the ease, speed and correctness of the integration in the system design are important aspects. A lot of information needs to be efficiently exchanged between companies using heterogeneous systems. For the module providers it is important to promote their modules, and provide complete and exact information openly. This benefits them, for example, by reducing the need for technical queries, as the vendors are the ones who provide the detailed information about their modules. An even greater benefit is that this will reduce the need for retyping and the re-creation of information, which will significantly reduce the effort across the supply chain, and the possibility of errors. The system integrator can collect exact and relevant information more quickly for their planned system. They are able to investigate and validate the different production scenarios more quickly, and make the right choices faster. The integrability of any proposed production system can be verified easily as the interface and process descriptions are all included.

The large end-user will be able to work with their own system architecture and can request information or a quotation about modules suitable for their production concept. This will support the level of harmonisation in the large companies. They already have a strict specification for equipment interfaces at the line or cell level of granularity. These usually have interface and equipment specifications which the supplier companies must comply with, which is already a step towards the aim of this concept.

The Emplacement Concept, production resource descriptions, and future integrated tools will enable a rapid and cost-effective reaction to dynamic market changes, reducing the efforts needed when switching between products and production quantities. Therefore, the following potential impacts can be expected: 1.) An increase in the number of product variations for existing production systems in an economical way; 2.) A reduction in the set-up and changeover times and costs for existing production systems, leading to a significant increase in production capacity; 3.) The emphasis on and support for the standardisation of communication protocols, data structures, and tool connectivity; 4.) A reduction in system downtime when a production module breaks down or fails in some way, through the increased exchangeability and interchangeability of resources and the ability to do automatic reconfiguration; 5.) Improvements in production capacity, with an optimal usage and configuration of production resources, and with an overall reduction of lead times and TTM; 6.) An increase in the service level with demand-driven production, both in terms of production volume and the variety of customer requests; 7.) Increased

sustainability and reduced energy and waste of natural resources through effective re-use of production resources; 8.) A reduction in overall energy consumption, with a more rational configuration of production resources enabling cost savings in the overall production and a lower carbon footprint for production plants, and 9.) An increase in the ability to tackle diversified markets or customer requirements in a cost-effective manner.

9.3 Future Work

There are a number of potential improvements that can be made to the concept, and some completely new directions for further research. The first improvement would be to standardise and refine the implementation format for HC in case needed. Two implementations, namely the Emplacement and BP, have places for optimisation streamlining, and the possibility for adding new features. However, these would require evaluation and feedback from industry, and anyway cannot be utilized until the next point has been achieved.

The next point is that more new tools are needed for utilising and supporting the description formats. This is important for the global acceptance and dissemination of the Emplacement Concept. This includes the implementation of import and export capabilities for existing tools and support for upcoming tools. These tools will be used for system design, deployment, commission and simulation. The development of editors for developing and maintaining the Emplacement, BP, and HC documents are included in this.

Thirdly, the implementation of heterogeneous control systems from the BP descriptions raises additional, practical considerations. The specific requirements of selected control system architectures and how their implementations need to be supported needs further analysis. According to this analysis, possible refinements to the descriptions for the control section need to be made. This could take the form of additional information about the available implementations of a specific skill within the module, as the same skill can be accessible for more than one technological solution and architecture. However, the immediate future objective is still to focus solely on the interface and the generalisability of the approach, so that orchestration of a control solution would be possible with this information. Duplicating the descriptions for control implementations should be avoided at all costs.

The fourth improvement relates to verification and validation. An (on-line) tool is needed for performing validation of the exchanged files. It should create a report of compliance and report possible issues within the files. Of particular importance would be the development of a methodology for the validation of the MDs against the AMDs. These are the phase four and five types of verification referred to in Ch. 6.3.

A fifth avenue to go down is to define mechanisms and processes for retrieving and updating the content of the Module Instance Descriptions that would prevent corruption or tampering with the information. There have to be procedures for authentication, security, and the updating of information from earlier records in a safe way. For example, when a module is removed from the system, the usage history of the module needs to be updated to include the usage hours from the previous operating period and the detachment date. This must be done in secure way without the risk of losing or corrupting data.

There are three completely new research directions which have become apparent while working on this thesis. The first relates to the development of a process taxonomy or ontology, or alternatively, to link an existing one. The process terminology, hierarchy, and parameters need to be standardized and enlarged. This includes agreeing on the names and IDs for different processes. Without such agreement, neither the product requirements for the production processes, nor the capabilities of the production resources, can be defined with a common terminology. If they do not use the same terminology, it is obvious that the two sides cannot be mapped to each other

during the various phases of production system design. This work has been already initiated in [65, 73, 75], but it needs completion and common agreement in the form of standardisation.

The second new direction to go in would be the development of common architectures and detailed interface standards for RMS and other modular production automation platforms, so that intra-cell modules can be easily connected. This is associated with the development of common (reference) architectures for RMS, which then can be followed. This point is, of course, tied in with the standard development. Normally the interface standards are not specific enough and leave room for different interpretations and ambiguity. For example, they do not provide a clear classification of the available classes and options. The exact location and orientation of an interface's origin is also undefined. The standards [22–24, 61, 131–133] are examples of where these requirements have been taken into account.

The third direction to go in is the automatic generation of GUIs. The resource description should include HMI screens and behaviour inside the generic type descriptions. These could then be compiled or interpreted into instantiations on screens at runtime. This would be very beneficial and would save a lot of time and effort in integration, commissioning, and use. However, the difficulty remains in selecting the right implementation strategy – meaning OS neutrality and technological solutions.

10 Conclusions

The increasing volatility in global economies, shortening product life cycles and the ever-increasing number of variants originating from product customisation are all placing new demands on production systems. Therefore, tomorrow's production systems require higher flexibility, adaptability, agility and reactivity. They must be quick and cost-efficient to set-up once a new product generation enters production. Reconfigurable modular systems can be seen as solution, in which the reconfiguration is a smooth and seamless operation when unexpected events such as resource breakdown occur. In any production system, resources always need to be added, removed or replaced. If all this could happen through auto-configuration, so no manual reprogramming is required, that would be a great leap forward. It is in that context that the objective of this thesis was conceived, i.e. to improve the design, implementation, and ramp-up process of modular production systems by developing of a method to define and describe reconfigurable production system resources. The requirements associated with this objective are described in Ch. 4.

The Emplacement Concept is the answer to this objective. It is a concept that supports the system design process of a Reconfigurable Manufacturing System (RMS) from readily available modules, and at the same time it promotes design-by-reuse. Ch. 5 provides an answer to the top level research question (Research Question 0): The Emplacement Concept consists of a Generic Model, which defines an abstract and layered data model for the production resources. The three defined layers are Abstract Module Description (AMD), Module Description (MD), and Module Instance Description (MID). The AMD targets harmonisation and exchangeability of production resources, by defining the mandatory set of interfaces and capabilities a resource must have. It can be further utilised for high level system design purposes. The MD describes the details of a production resource type and the MID is an information container for any individual instance of the production resource. The latter two are created by the module provider. The content of the MID is intended to be updated during the lifetime of the module, while the MD remains unchanged. It should be noted that the creation or change of an AMD is a more formal and controlled process and is implemented through, e.g. a harmonisation organisation.

Ch. 6 of this thesis proposed a practical implementation of the Generic Model. It consisted of Emplacement, Blueprint (BP), and History Container (HC) file formats, which correspond with the abstracted model and functions described by the Generic Model. The implementation was based on XML technologies, utilising the generic tool support available on the open market. The implementation was developed and verified by two case studies, which represented different kinds of modular production systems. All 25 different kinds of production modules were modelled with the BP and associated upper-layer descriptions - Emplacements. No major changes were needed to the structure of the data models when moving from one architecture or module to another, which provides proof of the generalisability of the proposed concept and its implementation. The evaluation of the results is presented in Ch. 9.

The main outcomes of this thesis are:

1. The development of the Emplacement Concept for delivering the necessary information about the production equipment modules. This information, in this form, can be utilised in

different reconfigurable system concepts like RMS, Holonic Manufacturing System (HMS), or Evolvable Production System (EPS).
2. Important production module information can be expressed in a common, formalized digital description format.
3. The description formats facilitate information exchange and resource integration between different manufacturers, system developments and design environments and executive frameworks on the shop floor.

The thesis discusses how the proposed Emplacement Concept can be utilised. Firstly, it presents a framework (Ch. 5.5), which provides a process model for production system design and commissioning. It identifies the associated tools and document formats used to link these tools together. Finally, it maps the Emplacement Concept to those tools and documents. In Ch. 7 there is a discussion of how the proposed Emplacement Concept and developed descriptions can be utilised in practical terms. This section mainly focused on system design, interfaces, and capabilities. A few initial tools for using and promoting the concept were presented.

However, the lack of suitable tools to utilize the concept leads us towards the final evaluation of the Emplacement Concept and the need for future work. At this time, there are only a few tools and the overall framework has clear gaps in the tool chain. In order to validate and prove the high-level objectives, these tools need to exist. Only then will it be possible to perform comparisons with current production system design procedures, and to show solid proof of the proposed concept and deliver the solution for the high-level objectives, which are: faster and better quality production system design, rapid ramp-up and reconfiguration, reduced Time to Market (TTM) and Time to Volume (TTV), easy system integration and module exchange, and improved sustainability of resources by re-use. From this point forward, the Emplacement Concept and associated formal descriptions are ready to be applied in practise and only need the development of the tools which have already been indicated as an area for future work (Ch. 9.3). We will go on.

References

[1] Tamio Arai, Y Aiyama, Yusuke Maeda, Masao Sugi, and J. Ota. "Agile Assembly System by "Plug and Produce"". In: *CIRP Annals - Manufacturing Technology* 49.1 (2000), pp. 1–4. DOI: 10.1016/S0007-8506(07)62883-2.

[2] AutomationML, ed. *AutomationML - The Glue for Seamless Automation Engineering*. Oct. 4, 2010. URL: https://www.automationml.org/o.red/uploads/dateien/1317722225-AutomationML_Flyer.pdf (visited on 06/14/2016).

[3] AutomationML. *AutomationML Specification, Part 1 - Architecture and General Requirements, Version 2.2*. Standard. July 2013, p. 84. URL: https://www.automationml.org/o.red/uploads/dateien/1375858464-AutomationML%20Whitepaper%20Part%201%20-%20AutomationML%20Architecture%20V2.2.pdf (visited on 07/24/2014).

[4] AutomationML. *AutomationML - home page*. Ed. by AutomationML. July 24, 2014. URL: http://www.automationml.org/ (visited on 06/14/2016).

[5] Martin John Baker. *Euclidean Space: Maths - Axis Angle*. 1998. URL: http://www.euclideanspace.com/maths/geometry/rotations/axisAngle/ (visited on 10/19/2009).

[6] José Barata. "Coalition Based Approach for Shop Floor Agility - A Multiagent Approach". PhD thesis. Universidade Nova de Lisboa, Portugal, 2003, p. 329.

[7] Matthias Bartelt, Adrian Schyja, and Bernd Kuhlenkötter. "More than a Mockup - SmartComponents: reusable fully functional virtual components from scratch". In: *Production Engineering* 8.6 (July 2014), pp. 727–735. ISSN: 0944-6524. DOI: 10.1007/s11740-014-0575-6. URL: http://link.springer.com/10.1007/s11740-014-0575-6.

[8] Gene Bellinger, Durval Castro, and Anthony Mills. *Data, Information, Knowledge, and Wisdom*. 2004. URL: http://www.systems-thinking.org/dikw/dikw.htm (visited on 06/29/2015).

[9] Lucienne T.M. Blessing and Amaresh Chakrabarti. *DRM, a Design Research Methodology*. London: Springer London, 2009, p. 411. ISBN: 978-1-84882-586-4. DOI: 10.1007/978-1-84882-587-1. URL: http://link.springer.com/10.1007/978-1-84882-587-1.

[10] Daniel P. Borches and Maarten G. Bonnema. "On the Origin of Evolvable Systems". In: *Tools and Methods of Competitive Engineering (TMCE)*. 2008, pp. 1–12.

[11] Vicente Botti and Adriana Giret. "Holonic Manufacturing Systems". In: *ANEMONA. A multi-agent methodology for Holonic Manufacturing*. Springer-Verlag London, 2008, pp. 7–20. ISBN: 978-1-84800-310-1. DOI: 10.1007/978-1-84800-310-1.

[12] Graeme Arthur Britton and Seppo Torvinen. *Design Synthesis: Integrated Product and Manufacturing System Design*. CRC Press, 2013, p. 380. ISBN: 9781439881644.

[13] Fabien Chiron and Khalid Kouiss. "Design of IEC 61131-3 function blocks using SysML". In: *2007 Mediterranean Conference on Control & Automation*. IEEE, July 2007, pp. 1–5. ISBN: 978-1-4244-1281-5. DOI: 10.1109/MED.2007.4433695. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4433695.

[14] James H Christensen, Thomas Strasser, Antonio Valentini, Valeriy Vyatkin, and Alois Zoitl. "The IEC 61499 Function Block Standard: Overview of the Second Edition". In: *ISA Automation Week* (2012). URL: http://www.holobloc.com/papers/iec61499/61499_ED2_OVW.pdf.

[15] C.I.R.P. *Dictionary of Production Engineering/Wörterbuch Der Fertigungstechnik/Dictionnaire Des Techniques De Production Mechanique Vol IV Assembly/Montage/Assemblage*. Ed. by C.I.R.P. The International Academy for Production Engineering. Springer Berlin Heidelberg, 2011. ISBN: 9783642120077. DOI: 10.1007/978-3-642-12007-7.

[16] Robert Cloutier, Gerrit Muller, Dinesh Verma, Roshanak Nilchiani, Eirik Hole, and Mary Bone. "The Concept of Reference Architectures". In: *Systems Engineering* 13.1 (2010), pp. 14–27. ISSN: 10981241. DOI: 10.1002/sys.20129. URL: http://doi.wiley.com/10.1002/sys.20129.

[17] Anne-Francoise Cutting-Decelle, Line Pouchard, Jean-jacques Michel, Robert Young, and Bishnu Das. "Utilising Standards Based Approaches to Information Sharing and Interoperability in Manufacturing Decision Support". In: *Flexible Automation and Intelligent Manufacturing (FAIM 2004)*. July 2004, p. 12. URL: www.nist.gov/msidlibrary/doc/FAIM2004_final.pdf.

[18] Anne-Francoise Cutting-Decelle, R.I.M. Young, Jean-Jacques Michel, Reyes Grangel, Julie Le Cardinal, and Jean-Pierre Bourey. "ISO 15531 MANDATE: A Product-process-resource based Approach for Managing Modularity in Production Management". In: *Concurrent Engineering* 15.2 (June 2007), pp. 217–235. ISSN: 1063-293X. DOI: 10.1177/1063293X07079329. URL: http://cer.sagepub.com/cgi/doi/10.1177/1063293X07079329.

[19] Rainer Drath, Arndt Lüder, Jörn Peschke, and Lorenz Hundt. "AutomationML - the glue for seamless automation engineering". In: *2008 IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, Sept. 2008, pp. 616–623. ISBN: 978-1-4244-1505-2. DOI: 10.1109/ETFA.2008.4638461. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4638461.

[20] EFFRA. *Factories of the future - Multi-annual roadmap for the contractual PPP under Horizon 2020*. European Commission, 2013, p. 136. ISBN: 9789279312380. DOI: 10.2777/29815. URL: http://www.effra.eu/index.php?option=com_content%5C&view=category%5C&layout=blog%5C&id=85%5C&Itemid=133.

[21] Hoda ElMaraghy. "Flexible and reconfigurable manufacturing systems paradigms". In: *International Journal of Flexible Manufacturing Systems* 17 (4 2005). DOI: 10.1007/s10696-006-9028-7, pp. 261–276. ISSN: 0920-6299. DOI: 10.1007/s10696-006-9028-7.

[22] EUPASS/AT, Göran Abbestam, Hans-Rudolf Helfer, and Niko Siltala. *EUPASS Carrier*. Standard EUPASS std-0001. EUPASS, Oct. 21, 2008, p. 23. URL: http://www.eas-env.org (visited on 10/15/2009).

[23] EUPASS/AT, Hans-Rudolf Helfer, and Niko Siltala. *EUPASS Workstation Framework*. Standard EUPASS std-0003. EUPASS, Oct. 1, 2008, p. 14. URL: http://www.eas-env.org (visited on 10/15/2009).

[24] EUPASS/AT, Hans-Rudolf Helfer, Niko Siltala, and Harrie Verstappen. *EUPASS Bay Interface*. Standard EUPASS std-0002. EUPASS, Dec. 16, 2008, p. 20. URL: http://www.eas-env.org (visited on 10/15/2009).

[25] EUPASS/AT, Niko Siltala, Ralf Heitmann, and Herbert Vollmer. *Blue Print Specification and Guideline*. Standard EUPASS std-0006. EUPASS, Oct. 2008, p. 40. URL: http://www.eas-env.org (visited on 10/15/2009).

[26] EUPASS/AT, Niko Siltala, Andreas Hofmann, Ulrich Gengenbach, Emanuele Travaini, and Herbert Vollmer. *Emplacement Specification and Guideline*. Standard EUPASS std-0004. EUPASS, Oct. 2008, p. 61. URL: http://www.eas-env.org (visited on 10/15/2009).

[27] EUPASS/WP3, Michel Jongerius, Göran Abbestam, Peter Schlaich, Gerd Hoppe, Gebhard Munz, Ulrich Moser, and Werner Kauwenberg. *D3.4.2: Module specification and test requirements (PV2, inclusive interfaces) (V3.0)*. Deliverable EUPASS D3.4.2. on Project CD. June 15, 2007, p. 305.

[28] EUPASS/WP3, Werner Kauwenberg, Göran Abbestam, Peter Schlaich, Gerd Hoppe, Gebhard Munz, and Ulrich Moser. *D3.5: Module designs for Platform Version 2 (V3.0)*. Deliverable EUPASS D3.5. on Project CD. Nov. 30, 2007, p. 136.

[29] EUPASS/WP3, Werner Kauwenberg, Göran Abbestam, Peter Schlaich, Gerd Hoppe, Gebhard Munz, and Ulrich Moser. *D3.7: Module prototypes for Platform Version 2 (V3.0)*. Deliverable EUPASS D3.7. on Project CD. Nov. 30, 2007, p. 145.

[30] EUPASS/WP4, Andreas Hofmann, Hans-Rudolf Helfer, Gavin Murray, and Gregor Burkhard. *D4.10e: EUPASS System Design*. Deliverable EUPASS D4.10e. on Project CD. Oct. 3, 2008, p. 25.

[31] EUPASS/WP4, Benjamin Loesch, Claude Rubattel, and Gregor Burkhard. *D4.14b: Planning, Evaluation and Validation of Platform Version 2*. Deliverable EUPASS D4.14b. on Project CD. Oct. 7, 2008, p. 87.

[32] *FDT: Technical Description*. Technical report. FDT Group AISBL, 2013. URL: http://www.fdtgroup.org/sites/default/files/pages/130326_Technical_Description_LR.pdf (visited on 07/14/2015).

[33] *FDT2.0 Technical Specification*. Technical Specification. FDT Group AISBL, 2012. URL: http://www.fdtgroup.org/sites/default/files/pages/FDT2-Specification-V1.00.00.00.zip (visited on 07/14/2015).

[34] Pedro Ferreira. "An Agent-Based Methodology for Modular Assembly Systems". PhD thesis. University of Nottingham, Apr. 2011, p. 257.

[35] Pedro Ferreira and Niels Lohse. "Configuration model for evolvable assembly systems". In: *CIRP Conference on Assembly Technologies and Systems (CATS) 2012* November (2012). Ed. by S.Editor Jack Hu, pp. 75–79. URL: http://cirp.me.engin.umich.edu/program.php.

[36] Pedro Ferreira, Niels Lohse, and Svetan Ratchev. “Multi-agent Architecture for Reconfiguration of Precision Modular Assembly Systems”. In: *Precision Assembly Technologies and Systems: 5th IFIP WG 5.5 International Precision Assembly Seminar, IPAS 2010*. Vol. 315/2010. 2010, pp. 247–254. DOI: 10.1007/978-3-642-11598-1_29.

[37] Regina Frei. “Self-Organisation in Evolvable Assembly Systems”. PhD thesis. Faculty of Science and Technology, Universidade Nova de Lisboa, Portugal, 2010, p. 273.

[38] Regina Frei, B Ferreira, and José Barata. “Dynamic Coalitions for Self-Organization in Evolvable Assembly Systems”. In: *CIRP Int. Conf. on Intelligent Computation in Manufacturing Engineering (ICME)*. 2008, p. 6. URL: http://scholar.google.com/scholar?hl=en%5C&btnG=Search%5C&q=intitle:Dynamic+Coalitions+for+Self-Organization+in+Evolvable+Assembly+Systems#0.

[39] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN: 0-201-63361-2.

[40] Tobias Gaugel, Matthias Bengel, and Dirk Malthan. “Building a mini-assembly system from a technology construction kit”. In: *Assembly Automation* 24.1 (2004), pp. 43–48. ISSN: 0144-5154. DOI: 10.1108/01445150410517174. URL: http://www.emeraldinsight.com/10.1108/01445150410517174.

[41] Carlos Gershenson. “Design and Control of Self-organizing Systems”. PhD thesis. Vrije Universiteit, Brussel, May 2007, p. 175. URL: http://scifunam.fisica.unam.mx/mir/copit/TS0002EN/TS0002EN.pdf.

[42] Juan M. González Calleros, Gerrit Meixner, Fabio Paternò, Jaroslav Pullmann, Dave Raggett, Daniel Schwabe, and Jean Vanderdonckt. *Model-Based UI XG Final Report*. 2010. URL: http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/ (visited on 07/24/2015).

[43] Robert Harms, Timo Fleschutz, and Günther Seliger. “Knowledge Based Approach to Assembly System Reuse”. In: *ASME 2008 9th Biennial Conference on Engineering Systems Design and Analysis (ESDA2008)*. 2008, pp. 295–302. ISBN: 978-0-7918-4835-7. DOI: 10.1115/ESDA2008-59247. URL: http://link.aip.org/link/abstract/ASMECP/v2008/i48357/p295/s1%5Cnhttp://link.aip.org/link/ASMECP/v2008/i48357/p295/s1%5C&Agg=doi.

[44] Riku Heikkilä, Eeva Järvenpää, and Reijo Tuokko. “Advances in the TUT Microfactory Concept Development”. In: *International Journal of Automation Technology* 4.2 (2010), pp. 117–126.

[45] Ralph L. Hollis and Jay Gowdy. “Miniature factories for precision assembly”. In: *International Workshop on Microfactories*. Citeseer, 1998, pp. 9–14.

[46] Ralph L. Hollis and Arthur E. Quaid. “An Architecture for Agile Assembly”. In: *Proc. Am. Soc. of Precision Engineering, 10th Annual Mtg. October 15-19, 1995.* Citeseer, Oct. 1995, pp. 1–4. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.8842%5C&rep=rep1%5C&type=pdf.

[47] Sebastian Horbach, Jörg Ackermann, Egon Müller, and Jens Schütze. “Building blocks for adaptable factory systems”. In: *Robotics and Computer-Integrated Manufacturing* 27 (Feb. 2011), pp. 735–740. DOI: 10.1016/j.rcim.2010.12.011.

[48] IEC. *Batch control - Part 1: Models and terminology*. Standard IEC 61512-1:1997. Aug. 26, 1997, p. 177. URL: http://www.iec.ch (visited on 07/15/2014).

[49] IEC. *Function blocks (FB) for process control - Part 2: Specification of FB concept*. Standard IEC 61804-2:2006. Sept. 2006, p. 126. URL: http://www.iec.ch (visited on 07/23/2015).

[50] IEC. *Function blocks - Part 1: Architecture*. Standard IEC 61499-1(ed.2.0):2012. Nov. 2012, p. 245. URL: http://www.iec.ch (visited on 11/11/2014).

[51] IEC. *Enterprise-control system integration - Part 1: Models and terminology*. Standard IEC 62264-1:2013. May 28, 2013, p. 154. URL: http://www.iec.ch (visited on 07/15/2014).

[52] IEC. *Programmable controllers - Part 3: Programming languages*. Standard IEC 61131-3(ed.3.0):2013. IEC, Feb. 2013, p. 464. URL: http://www.iec.ch (visited on 11/11/2014).

[53] IEC. *Engineering data exchange format for use in industrial automation systems engineering - Automation markup language - Part 1: Architecture and general requirements*. Standard IEC 62714-1:2014. June 26, 2014, p. 170. URL: http://www.iec.ch (visited on 07/24/2014).

[54] IEC. *Function Blocks (FB) for process control - Part 3: Electronic Device Description Language (EDDL) syntax and semantics*. Standard IEC 61804-3:2015. June 2015, p. 694. URL: http://www.iec.ch (visited on 07/23/2015).

[55] ISO. *Industrial automation systems and integration - Open systems application integration framework - Part1:Generic reference description*. Standard ISO 15745-1:2003. Mar. 2003, p. 40. URL: http://www.iso.org (visited on 01/09/2008).

[56] ISO. *Industrial automation systems and integration - Open systems application integration framework - Part3:Reference description for IEC 61158-based control systems*. Standard ISO 15745-3:2003. Nov. 2003, p. 269. URL: http://www.iso.org (visited on 01/09/2008).

[57] ISO. *Industrial automation systems and integration – Process specification language – Part 1: Overview and basic principles*. Standard ISO 18629-1:2004. Nov. 2004, p. 37. URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=35431 (visited on 12/01/2014).

[58] ISO. *Industrial automation systems and integration – Industrial manufacturing management data: Resources usage management – Part 32: Conceptual model for resources usage management data*. Standard ISO 15531-32:2005. Oct. 2005, p. 37. URL: http://www.iso.org (visited on 08/13/2015).

[59] ISO. *Information technology – Document Schema Definition Languages (DSDL) – Part 3: Rule-based validation – Schematron*. Standard ISO 19757-3:2006. Jan. 2006, p. 30. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip (visited on 09/22/2015).

[60] ISO. *Quantities and units – Part 1: General*. Standard ISO 80000-1:2009. Nov. 2009, p. 41. URL: http://www.iso.org/iso/catalogue_detail?csnumber=30669 (visited on 10/06/2015).

[61] ISO. *Production equipment for microsystems – Interface between end effector and handling system*. Standard ISO 29262:2011. Feb. 17, 2011, p. 17. URL: http://www.iso.org (visited on 06/30/2014).

[62] Akio Ito. "Device configuration software development on EDDL and FDT/DTM environment". In: *2008 SICE Annual Conference*. IEEE, Aug. 2008, pp. 929–932. ISBN: 978-4-907764-30-2. DOI: 10.1109/SICE.2008.4654788.

[63] Atul Jain, D. A. Vera, and R. Harrison. “Virtual commissioning of modular automation systems”. In: *10th IFAC Workshop on Intelligent Manufacturing Systems (2010)*. Vol. 10. 1. 2010, pp. 72–77. DOI: 10.3182/20100701-2-PT-4011.00014. URL: http://www.ifac-papersonline.net/Detailed/43344.html.

[64] Marcin Jamro and Bartosz Trybus. “An approach to SysML modeling of IEC 61131-3 control software”. In: *2013 18th International Conference on Methods & Models in Automation & Robotics (MMAR)*. IEEE, Aug. 2013, pp. 217–222. ISBN: 978-1-4673-5508-7. DOI: 10.1109/MMAR.2013.6669909. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6669909.

[65] Eeva Järvenpää. “Capability-based Adaptation of Production Systems in a Changing Environment”. PhD thesis. Tampere University of Technology, Nov. 2012, p. 190. ISBN: 978-952-15-2928-3. URL: http://urn.fi/URN:ISBN:978-952-15-2940-5.

[66] Reza N. Jazar. *Theory of Applied Robotics: Kinematics, Dynamics, and Control*. Springer US, 2010, p. 883. ISBN: 978-1-4419-1749-2. DOI: 10.1007/978-1-4419-1750-8. URL: http://link.springer.com/10.1007/978-1-4419-1750-8.

[67] John Johansen, Ole Madsen, Henrik Valentin Jensen, and Anders Vestergaard. *Manufacturing 2025: Five future scenarios for Danish manufacturing companies*. 2010, p. 35. ISBN: 87-91831-20-2.

[68] John Johansen and Anders Vestergaard. “Manufacturing 2025”. In: *Manufuture presentation*. Manufuture.DK, 2010, p. 15.

[69] Yoram Koren. “General RMS characteristics. Comparison with dedicated and flexible systems”. In: *Reconfigurable Manufacturing Systems and Transformable Factories*. Ed. by Anatoli I. Dashchenko. Springer, 2006. Chap. 3, pp. 27–45. ISBN: 3-540-29391-4.

[70] Yoram Koren, U. Heisel, Francesco Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel. “Reconfigurable Manufacturing Systems”. In: *CIRP Annals - Manufacturing Technology* 48.2 (1999), pp. 527–540. DOI: 10.1016/S0007-8506(07)63232-6.

[71] Microdynamic Systems Laboratory. *An Architecture for Agile Assembly*. Ed. by Microdynamic Systems Laboratory. Carnegie Mellon University / Robotics Institute / Microdynamic Systems Laboratory. Sept. 26, 2013. URL: http://www.msl.ri.cmu.edu/projects/minifactory/ (visited on 06/12/2014).

[72] Robert G. Landers, J Ruan, and Frank Liou. “Reconfigurable manufacturing equipment”. In: *Reconfigurable Manufacturing Systems and Transformable Factories*. Ed. by Anatoli I.Editor Dashchenko. Springer, 2006. Chap. 6, pp. 79–110. ISBN: 3-540-29391-4.

[73] Minna Lanz. “Logical and semantic foundations of knowledge representation for assembly and manufacturing processes”. PhD thesis. Tampere University of Technology, June 2010, p. 137. ISBN: 1459-2045; 978-952-15-2393-9. URL: http://URN.fi/URN:NBN:fi:tty-201006291195.

[74] Anthony Liew. “Understanding Data, Information, Knowledge And Their Inter-Relationships”. In: *Journal of Knowledge Management Practice* 8.2 (2007), pp. 1–7. URL: http://www.tlainc.com/articl134.htm.

[75] Niels Lohse. “Towards an Ontology Framework for The Integrated Design of Modular Assembly Systems”. PhD thesis. University of Nottingham, May 2006, p. 245.

[76] Niels Lohse, Hitendra Hirani, and Svetan Ratchev. “Equipment ontology for modular reconfigurable assembly systems”. In: *International Journal of Flexible Manufacturing Systems* 17.4 (Oct. 2006), pp. 301–314. ISSN: 0920-6299. DOI: 10.1007/s10696-006-9030-0. URL: http://www.springerlink.com/index/10.1007/s10696-006-9030-0.

[77] Niels Lohse, Hitendra Hirani, Svetan Ratchev, and Michele Turitto. “An ontology for the definition and validation of assembly processes for evolvable assembly systems”. In: *(ISATP 2005). The 6th IEEE International Symposium on Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005.* (2005), pp. 242–247. DOI: 10.1109/ISATP.2005.1511480. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1511480.

[78] Antonio Maffei. “Characterisation of the Business Models for Innovative, Non-Mature Production Automation Technology”. PhD thesis. The Royal Institute of Technology (KTH), 2012, p. 320. ISBN: 978-91-7501-588-0. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-105111.

[79] Antonio Maffei, Kerstin Dencker, Marcus Bjelkemyr, and Mauro Onori. “From Flexibility to Evolvability: Ways to achieve self-reconfigurability and full autonomy”. In: *International IFAC symposium on robot control (SYROCO), Gifu, Japan*. 2009, pp. 171–176.

[80] Tiziano Maraldo, Mauro Onori, José Barata, and Daniel Semere. “Evolvable Assembly Systems: Clarifications and Developments to Date”. In: *Proceedings of the CIRP / IWES 6th International Workshop on Emergent Synthesis*. 2006. URL: http://oa.uninova.pt/392/.

[81] Paul G. Maropoulos and Dariuz Ceglarek. “Design verification and validation in product lifecycle”. In: *CIRP Annals - Manufacturing Technology* 59.2 (Jan. 2010), pp. 740–759. ISSN: 00078506. DOI: 10.1016/j.cirp.2010.05.005. URL: http://linkinghub.elsevier.com/retrieve/pii/S0007850610001927.

[82] Paul G. Maropoulos, K.R. McKay, and D.G. Bramall. “Resource-Aware Aggregate Planning for the Distributed Manufacturing Enterprise”. In: *CIRP Annals - Manufacturing Technology* 51.1 (Jan. 2002), pp. 363–366. ISSN: 00078506. DOI: 10.1016/S0007-8506(07)61537-6. URL: http://opus.bath.ac.uk/2655/.

[83] Jose L. Martinez Lastra. “Reference Mechatronic Architecture for Actor-based Assembly Systems”. PhD thesis. Tampere University of Technology, 2004, p. 140.

[84] Mostafa G Mehrabi, Galip Ulsoy, and Yoram Koren. “Reconfigurable manufacturing systems: Key to future manufacturing”. In: *Journal of Intelligent Manufacturing* 11.4 (2000), pp. 403–419. DOI: 10.1023/A:1008930403506.

[85] Oliver Merget. “The electronic integration of field devices”. In: *Computing Control Engineering Journal* 14.5 (Nov. 2003), pp. 22–23. ISSN: 0956-3385.

[86] Paul Miller. “Interoperability. What is it and Why should I want it?” In: *Ariadne* 24 (June 2000). URL: http://www.ariadne.ac.uk/issue24/interoperability/ (visited on 03/06/2012).

[87] Ali Mir Farooq, Manuel A. Pérez-Quiñones, Marc Abrams, and Eric Shell. “Building Multi-Platform User Interfaces with UIML”. In: *Computer-Aided Design of User Interfaces III*. March. Springer Netherlands, 2002, pp. 255–266. ISBN: 978-94-010-3915-4 (print) 978-94-010-0421-3 (online). DOI: 10.1007/978-94-010-0421-3_22. URL: http://arxiv.org/abs/cs/0111024.

[88] A. Molina, C. A. Rodriguez, H. Ahuett, J. A. Cortés, M. Ramírez, G. Jiménez, and S. Martinez. “Next-generation manufacturing systems: Key research issues in developing and integrating reconfigurable and intelligent machines”. In: *International Journal of Computer Integrated Manufacturing* 18.7 (Oct. 2005), pp. 525–536. DOI: 10.1080/09511920500069622.

[89] Y. M. Moon. “Reconfigurable Machine Tool Design”. In: *Reconfigurable Manufacturing Systems and Transformable Factories*. Ed. by Anatoli I.Editor Dashchenko. Springer, 2006. Chap. 7, pp. 111–139. ISBN: 3-540-29391-4.

[90] Ram Mudambi. “Offshoring: Economic Geography and the Multinational Firm”. In: *Journal of International Business Studies* 38.1 (Jan. 2007), p. 206. DOI: 10.1057/palgrave.jibs.8400253.

[91] Ram Mudambi and Markus Venzin. “The Strategic Nexus of Offshoring and Outsourcing Decisions”. In: *Journal of Management Studies* 47.8 (Dec. 2010), pp. 1510–1533. DOI: 10.1111/j.1467-6486.2010.00947.x.

[92] US Navy. *Open Systems Architecture*. 2013. URL: http://www.navy.mil/navydata/fact_display.asp?cid=2100&tid=450&ct=2 (visited on 05/27/2016).

[93] Pedro Miguel Neves. “Reconfiguration Methodology to improve the agility and sustainability of Plug and Produce Systems”. PhD thesis. The Royal Institute of Technology (KTH), 2016, p. 188. ISBN: 978-91-7595-947-4. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-186177.

[94] NIST. *Secure Hash Standard (SHS)*. Standard FIPS PUB 180-4. Aug. 1, 2015, p. 31. DOI: 10.6028/NIST.FIPS.180-4. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (visited on 09/22/2015).

[95] Matti Öhman, Jouko Kalmari, and Arto Visala. “XML Based Graphical User Interface Editor and Runtime Parser for ISO 11783 Machine Automation System”. In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*. Ed. by ChungEditor Myung. Vol. 17. 1 PART 1. July 2008, pp. 1578–1583. ISBN: 9783902661005. DOI: 10.3182/20080706-5-KR-1001.00269. URL: http://www.ifac-papersonline.net/Detailed/35995.html.

[96] OMG. *Service oriented architecture Modeling Language (SoaML)*. Standard SoaML v1.0.1. May 10, 2012. URL: http://www.omg.org/spec/SoaML/1.0.1/ (visited on 06/19/2016).

[97] OMG. *Systems Modeling Language (SysML)*. Standard SysML v1.3. June 1, 2012, p. 250. URL: http://www.omg.org/spec/SysML/1.3/ (visited on 11/18/2014).

[98] Mauro Onori. “Evolvable Assembly Systems - A New Paradigm?” In: *Proceedings of the 33rd International Symposium on Robotics (ISR)*. Oct. 2002, pp. 617–622.

[99] Mauro Onori, José Barata, and Regina Frei. “Evolvable Assembly Systems Basic Principles”. In: *Information Technology For Balanced Manufacturing Systems*. Ed. by WeimingEditor Shen. IFIP Inter. Vol. 220. Springer Boston, 2006. Chap. 34, pp. 317–328. DOI: 10.1007/978-0-387-36594-7_34.

[100] Mauro Onori, Jose L. Martinez Lastra, and Henric Alsterman. “Evolvable Assembly System Platforms - Definitions, Approaches and Requirements”. In: *Proceedings of the International Precision Assembly Seminar IPAS’2004*. 2004, p. 8.

[101] Mauro Onori, Niels Lohse, José Barata, and Christoph Hanisch. “The IDEAS project: plug & produce at shop-floor level”. In: *Assembly Automation* 32.2 (2012), pp. 124–134. ISSN: 0144-5154. DOI: 10.1108/01445151211212280. URL: http://www.emeraldinsight.com/10.1108/01445151211212280.

[102] Jarkko Pakkanen. “BROWNFIELD PROCESS - A Method for Rationalisation of Existing Product Variety towards a Modular Product Family”. PhD thesis. Tampere University of Technology, May 2015, p. 283. ISBN: 978-952-15-3524-6. URL: http://URN.fi/URN:ISBN:978-952-15-3537-6.

[103] Rodrigo Palucci Pantoni and Dennis Brandão. “Developing and implementing an open and non-proprietary device description for FOUNDATION fieldbus based on software standards”. In: *Computer Standards & Interfaces* 31.2 (Feb. 2009), pp. 504–514. DOI: 10.1016/j.csi.2008.06.008.

[104] Rodrigo Palucci Pantoni, Luis Carlos Passarini, and Dennis Brandão. “Developing and implementing an open and non-proprietary device description for fieldbus devices based on software standards”. In: *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*. June 2007, pp. 1887–1892. DOI: 10.1109/ISIE.2007.4374895.

[105] Nikolaos Papakonstantinou and Seppo Sierla. “Generating an Object Oriented IEC 61131-3 software product line architecture from SysML”. In: *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, Sept. 2013, pp. 1–8. ISBN: 978-1-4799-0864-6. DOI: 10.1109/ETFA.2013.6648057. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6648057.

[106] Julius Pfrommer, Miriam Schleipen, and Jurgen Beyerer. “PPRS: Production skills and their relation to product, process, and resource”. In: *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, Sept. 2013, pp. 1–4. ISBN: 978-1-4799-0864-6. DOI: 10.1109/ETFA.2013.6648114. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6648114.

[107] Angel Puerta and Jacob Eisenstein. “XIML: A Universal Language for User Interfaces”. In: *IUI ’02: Proceedings of the 7th international conference on Intelligent user interfaces*. 2002, pp. 214–215. URL: http://www.ximl.org/documents/XimlWhitePaper.pdf (visited on 07/20/2015).

[108] Sudarsan Rachuri, Eswaran Subrahmanian, Abdelaziz Bouras, Steven J. Fenves, Sebti Foufou, and Ram D. Sriram. “Information sharing and exchange in the context of product lifecycle management: Role of standards”. In: *Computer-Aided Design* 40.7 (July 2008), pp. 789–800. ISSN: 00104485. DOI: 10.1016/j.cad.2007.06.012. URL: http://linkinghub.elsevier.com/retrieve/pii/S0010448507001613.

[109] Hubert K. Rampersad. “Concentric design of robotic assembly systems”. In: *Journal of Manufacturing Systems* 14.4 (1995), pp. 230–243. DOI: 10.1016/0278-6125(95)98876-8.

[110] Hubert K. Rampersad. “Integrated and assembly oriented product design”. In: *Integrated Manufacturing Systems* 7.6 (1996), pp. 5–15. DOI: 10.1108/09576069610151130.

[111] *Reference Architecture Description*. Tech. rep. US Dept. of Defence / Office of the DoD CIO, 2010. URL: http://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf (visited on 05/25/2016).

[112] Gunther Reinhart and Georg Wünsch. “Economic application of virtual commissioning to mechatronic production systems”. In: *Production Engineering* 1.4 (2007), pp. 371–379. ISSN: 0944-6524. DOI: 10.1007/s11740-007-0066-0.

[113] Luis Ribeiro. “Diagnosis in Evolvable Production Systems”. PhD thesis. Universidade Nova de Lisboa, Jan. 2012, p. 231.

[114] *ROS - Unified Robot Description Format (URDF)*. Open Source Robotics Foundation. 2014. URL: http://wiki.ros.org/urdf (visited on 11/21/2014).

[115] Jennifer Rowley. “The wisdom hierarchy: representations of the DIKW hierarchy”. In: *Journal of Information Science* 33.2 (2007), pp. 163–180. ISSN: 0165-5515. DOI: 10.1177/0165551506070706. URL: http://hdl.handle.net/2173/90651.

[116] E. Sandin, P. Gröndahl, and Mauro Onori. “A process-oriented product design based on an assembly module platform”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. Vol. 4. May. IEEE, 2002, pp. 4179–4184. ISBN: 0-7803-7272-7. DOI: 10.1109/ROBOT.2002.1014406. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1014406.

[117] Adrian Schyja, Matthias Bartelt, and Bernd Kuhlenkötter. “From Conception Phase up to Virtual Verification Using AutomationML”. In: *Procedia CIRP* 23 (2014), pp. 171–177. ISSN: 22128271. DOI: 10.1016/j.procir.2014.10.067.

[118] Daniel Semere, Mauro Onori, Antonio Maffei, and Raphael Adamietz. “Evolvable assembly systems: Coping with variations through evolution”. In: *Assembly Automation* 28.2 (2008). Compilation and indexing terms, Copyright 2008 Elsevier Inc., pp. 126–133. DOI: 10.1108/01445150810863707.

[119] A.I. Shabaka and Hoda A. ElMaraghy. “Generation of machine configurations based on product features”. In: *International Journal of Computer Integrated Manufacturing* 20.4 (June 2007), pp. 355–369. ISSN: 0951-192X. DOI: 10.1080/09511920600740627.

[120] Niko Siltala, Riku Heikkilä, Asser Vuola, and Reijo Tuokko. “Architectures and Interfaces for a Micro Factory Concept”. In: *5th IFIP WG 5.5, International Precision Assembly Seminar, IPAS 2010*. Ed. by Svetan Ratcev. Vol. 315/2010. Ifip International Federation For Information Processing, 2010, pp. 293–300. DOI: 10.1007/978-3-642-11598-1_34.

[121] Niko Siltala, Andreas F. Hofmann, Reijo Tuokko, and Georg Bretthauer. “Emplacement and blue print - An approach to handle and describe modules for evolvable assembly systems”. In: *9th IFAC Symposium on Robot Control, SYROCO ’09, Nagaragawa Convention Center, September 9-12, 2009, Gifu, Japan*. Ed. by Hashimoto Hideki. 2009, pp. 183–188.

[122] Niko Siltala, Timo Prusi, Asser Vuola, Riku Heikkilä, and Reijo Tuokko. “Modular Microfactory System for Gas Sensor Assembly”. In: *Proceedings of International Symposium on Assembly and Manufacturing (ISAM2011)*. IEEE, May 2011, p. 6. DOI: 10.1109/ISAM.2011.5942333.

[123] Niko Siltala and Reijo Tuokko. “Emplacement and blue print - An approach to support rapid design and deployment of assembly systems”. In: *Proceedings of The International 3rd Swedish Production Symposium, SPS’09, 2-3 December, 2009, Göteborg, Sweden*. Ed. by B. G. Rosen. Sweden: The Swedish Production Academy, 2009, pp. 58–64. ISBN: 978-91-633-6006-0.

[124] Niko Siltala and Reijo Tuokko. “Emplacement and blue print - Electronic module description supporting evolvable assembly systems design, deployment and execution”. In: *Proceedings of the 6th CIRP-sponsored International Conference on Digital Enterprise Technology (DET), December 14-16, 2009, Hongkong. Advances in Soft Computing*. Ed. by G. Q. Huang, K. L. Mak, and P. G. Maropoulos. Vol. 66. Dec. 2009, pp. 773–788. ISBN: 1867-5662. DOI: 10.1007/978-3-642-10430-5_60.

[125] Niko Siltala and Reijo Tuokko. “Use of electronic module descriptions for modular and reconfigurable assembly systems”. In: *Proceedings of 2009 IEEE International Symposium on Assembly and Manufacturing (ISAM 2009), 17-20 November, 2009, Suwon, Korea*. IEEE, Nov. 2009, pp. 214–219. ISBN: 978-1-4244-4628-5. DOI: 10.1109/ISAM.2009.5376903.

[126] Niko Siltala and Reijo Tuokko. “A web tool supporting the management and use of electronic module descriptions for Evolvable Production Systems”. In: *Proceedings of 2010 IEEE International Symposium on Industrial Electronics (ISIE 2010), 4-7 July, 2010, Bari, Italy*. 2010, pp. 2641–2646. ISBN: 978-1-4244-6390-9. DOI: 10.1109/ISIE.2010.5637859.

[127] Rene Simon, Christian Diedrich, Matthias Riedl, and Mario Thron. “Field device integration”. In: *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570)*. Vol. 3. Ieee, 2001, pp. 150–155. ISBN: 0-7803-7090-2. DOI: 10.1109/ISIE.2001.931772.

[128] Nathalie Souchon and Jean Vanderdonckt. “A Review of XML-compliant User Interface Description Languages”. In: *Interactive Systems. Design, Specification, and Verification*. Ed. by J.A. Jorge, N. Jardim Nunes, and J.Editors Falcão E Cunha. Springer, 2003, pp. 377–391. DOI: 10.1007/978-3-540-39929-2_26. URL: http://www.springerlink.com/index/NFTBQFKUTXMJ7YN5.pdf.

[129] Richard Stevens, Peter Brook, Ken Jackson, and Stuart Arnold. *Systems Engineering: Coping with Complexity*. Prentice Hall Europe, 1998, p. 392. ISBN: 978-0-13-095085-7.

[130] Pedro Mendes. *Blue Print Editor software*. http://www.uninova.pt. Software. Software for editing BluePrint files. UNINOVA, 2009.

[131] TUT microfactory team: and Niko Siltala. *TUT uF - Data Exchange*. Standard TUT-uF std-0003. June 2, 2010, p. 11. URL: http://emplacementws.tte.tut.fi/EmplacementWS/ (visited on 08/01/2014).

[132] TUT microfactory team: Niko Siltala, Riku Heikkilä, and Asser Vuola. *TUT uF - Cell Interface*. Standard TUT-uF std-0001. June 16, 2010, p. 20. URL: http://emplacementws.tte.tut.fi/EmplacementWS/ (visited on 08/01/2014).

[133] TUT microfactory team: Niko Siltala, Riku Heikkilä, and Asser Vuola. *TUT uF - Process Module Interface*. Standard TUT-uF std-0002. June 16, 2010, p. 14. URL: http://emplacementws.tte.tut.fi/EmplacementWS/ (visited on 08/01/2014).

[134] Kleanthis Thramboulidis. “Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation”. In: *Journal of Software Engineering and Applications* 04.04 (2011), pp. 217–226. ISSN: 1945-3116. DOI: 10.4236/jsea.2011.44024. URL: http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/jsea.2011.44024.

[135] Kleanthis Thramboulidis and Andrea Buda. “3+1 SysML view model for IEC61499 Function Block control systems”. In: *2010 8th IEEE International Conference on Industrial Informatics*. IEEE, July 2010, pp. 175–180. ISBN: 978-1-4244-7298-7. DOI: 10.1109/INDIN.2010.5549440. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5549440.

[136] *User Interface Markup Language (UIML) Specification*. Standard v4.0. May 1, 2009. URL: http://docs.oasis-open.org/uiml/v4.0/uiml-4.0.pdf (visited on 07/20/2015).

[137] Hendrik Van Brussel. “Holonic Manufacturing Systems, the vision matching the problem”. In: *Proc. of First European Conf. on Holonic Manufacturing Systems*. Dec. 1994.

[138] Hendrik Van Brussel, Paul Valckenaers, Luc Bongaerts, and Jo Wyns. “Architectural and System Design Issues in Holonic Manufacturing Systems”. In: *3rd IFAC Workshop on Intelligent Manufacturing Systems (IMS)*. Nov. 1995, pp. 1–6.

[139] Hendrik Van Brussel, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. “Reference architecture for holonic manufacturing systems: PROSA”. In: *Computers in Industry* 37 (3 Nov. 1998), pp. 255–276. ISSN: 0166-3615. DOI: 10.1016/S0166-3615(98)00102-X. URL: http://portal.acm.org/citation.cfm?id=308344.308352.

[140] Jean Vanderdonckt, Quentin Limbourg, Benjamin Michotte, Laurent Bouillon, Daniela Trevisan, and Murielle Florins. “USIXML : a User Interface Description Language for Specifying Multimodal User Interfaces”. In: *Proceedings of W3C Workshop on Multimodal Interaction WMI*. 2004, pp. 19–20. URL: http://www.auto.2516.org/2004/02/mmi-workshop/vanderdonckt-louvain.pdf (visited on 07/20/2015).

[141] Parag Vichare, Aydin Nassehi, Sanjeev Kumar, and Stephen T. Newman. “A Unified Manufacturing Resource Model for representing CNC machining systems”. In: *Robotics and Computer-Integrated Manufacturing* 25.6 (Dec. 2009), pp. 999–1007. ISSN: 07365845. DOI: 10.1016/j.rcim.2009.04.014. URL: http://www.sciencedirect.com/science/article/pii/S073658450900043X.

[142] Parag Vichare, Aydin Nassehi, and Stephen T. Newman. “A unified manufacturing resource model for representation of computerized numerically controlled machine tools”. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 223.5 (May 2009), pp. 463–483. ISSN: 0954-4054. DOI: 10.1243/09544054JEM1363. URL: http://pib.sagepub.com/lookup/doi/10.1243/09544054JEM1363.

[143] W3C. *XSL Transformations (XSLT) - Technical Recommendation - v1.0*. Standard. Nov. 16, 1999. URL: http://www.w3.org/TR/xslt (visited on 06/07/2015).

[144] W3C. *Service Modeling Language Interchange Format (SML-IF)*. Standard SML-IF v1.1. May 12, 2009. URL: https://www.w3.org/TR/sml-if/ (visited on 06/19/2016).

[145] W3C. *Service Modeling Language (SML)*. Standard SML v1.1. May 12, 2009. URL: https://www.w3.org/TR/sml/ (visited on 06/19/2016).

[146] W3C. *XML Path Language (XPath) - Technical Recommendation - v3.1*. Standard. Dec. 18, 2014. URL: http://www.w3.org/TR/xpath-3/ (visited on 10/09/2015).

[147] *businessdictionary.com - Online Dictionary*. WebFinance Inc. 2015. URL: http://www.businessdictionary.com/definition/%3CTERM%3E.html (visited on 06/30/2015).

[148] *Merriam-Webster Online Dictionary*. Encyclopedia Britannica. 2015. URL: http://www.merriam-webster.com/dictionary/%3CTERM%3E (visited on 06/16/2015).

[149] ASM Assembly Systems GmbH & Co.KG, ed. *Siplace web page - SMT Placement Systems - Overview of SIPLACE X-Series*. ASM Assembly Systems GmbH & Co.KG. 2015. URL: http://www.siplace.com/en/placement-systems/x-series (visited on 10/21/2015).

[150] IMA Automation, ed. *IMA Automation web page - Assembly Systems*. IMA Automation. 2012. URL: http://www.ima-automation.de/en/customer-solutions/assembly-systems.html (visited on 03/29/2012).

[151] Mikron, ed. *Mikron web page - Products - Mikron EcoLine*. Mikron. 2012. URL: http://www.mikron.com/en/mikron-automation/solutions/products/mikron-ecoline/ (visited on 03/29/2012).

[152] *BihlerNC.com - Modular and reconfigurable NC machines*. Otto Bihler Maschinenfabrik GmbH & Co. KG. 2015. URL: http://www.bihlernc.com/maschinen/uebersicht_eng.htm (visited on 10/20/2015).

[153] OMG. *OMG Systems Modeling Language (SysML)*. OMG. 2014. URL: http://www.omgsysml.org/ (visited on 11/18/2014).

[154] *Field Device Configuration Markup Language*. 2009. URL: http://www.fdcml.org/ (visited on 10/17/2011).

[155] *Electronic Device Description Language (EDDL)*. 2015. URL: http://www.eddl.org/ (visited on 06/05/2015).

[156] *FDT Group - FDT Technology*. Field Device Tool(FDT) Group. 2015. URL: http://www.fdtgroup.org/fdt-technology-what-it (visited on 06/07/2015).

[157] NIST, ed. *Process Specification Language (PSL) – A Few PSL Basics...* NIST. 2008. URL: http://www.mel.nist.gov/psl/ (visited on 12/01/2014).

[158] OPC Foundation. *OPC Unified Architecture*. OPC Foundation. 2014. URL: https://opcfoundation.org/about/opc-technologies/opc-ua/ (visited on 11/06/2014).

[159] PLCOpen, ed. *XML Formats for IEC 61131-3. Version 2.0. Technical Paper. PLCopen Technical Committee 6*. PLCopen. 2008. URL: http://www.plcopen.org/pages/tc6_xml/index.htm (visited on 04/22/2009).

[160] *Vorto*. Vorto community. 2016. URL: http://www.eclipse.org/vorto/documentation/overview/introduction.html (visited on 05/19/2016).

[161] *Robot Operating System (ROS) - Core components*. Open Source Robotics Foundation. 2014. URL: http://www.ros.org/core-components/ (visited on 11/21/2014).

[162] ROS Industrial Consortium. *Robot Operating System (ROS) - Industrial*. ROS Industrial Consortium. 2014. URL: http://rosindustrial.org/ (visited on 11/21/2014).

[163] *User Interface Markup Language (UIML)*. OASIS. 2015. URL: http://www.uiml.org/ (visited on 07/20/2015).

[164] *USer Interface eXtensible Markup Language (UsiXML)*. UsiXML Org. 2015. URL: http://www.usixml.org/ (visited on 07/20/2015).

[165] *ISOAgLib Open-Source programming library for an ISOBUS-system (ISO 11783)*. OSB AG. 2015. URL: http://www.isoaglib.com/en/ (visited on 07/20/2015).

[166] *Fachhochschule Nordwestschweiz (FHNW)*. FHNW. 2015. URL: http://www.fhnw.ch/ (visited on 06/07/2015).

[167] Angus Stevenson, ed. *Oxford Dictionary of English (3 ed.)* 2013. URL: http://www.oxfordreference.com/view/10.1093/acref/9780199571123.001.0001/acref-9780199571123 (visited on 07/15/2014).

[168] IEEE, ed. *IEEE Glossary*. IEEE. 2012. URL: http://www.ieee.org/education_careers/education/standards/standards_glossary.html (visited on 02/28/2012).

[169] TechTarget, ed. *Definition of Interoperability*. http://whatis.techtarget.com/. 2012. URL: http://searchsoa.techtarget.com/definition/interoperability (visited on 02/28/2012).

[170] *The Open Source Definition*. Open Source Initiative. 2015. URL: http://opensource.org/docs/osd (visited on 06/07/2015).

[171] Margaret Rouse. *Definition of technical terms*. Ed. by TechTarget. 2015. URL: http://searchsoftwarequality.techtarget.com/definition/%3Citem%3E (visited on 09/08/2015).

[172] TechTarget, ed. *Definition of Interface*. http://whatis.techtarget.com/. 2012. URL: http://searchcio-midmarket.techtarget.com/definition/interface (visited on 02/28/2012).

[173] Niko Siltala. *Emplacement Web Service*. TUT/MEI. 2009. URL: http://resourcedescription.tut.fi/EmplacementWS/ (visited on 05/27/2016).

[174] EUPASS consortia, ed. *Evolvable Ultra-Precision Assembly SystemS - EU 6th Framework Program project*. EUPASS. 2008. URL: http://www.eupass-fp6.org (visited on 06/15/2009).

[175] *Schematron developer forum*. 2014. URL: http://www.schematron.com/ (visited on 06/01/2015).

[176] *XML Spy*. Altova. 2015. URL: http://www.altova.com/xmlspy.html (visited on 11/11/2015).

[177] Riku Heikkilä. *TUT microfactory*. Web page. 2005. URL: http://www.tut.fi/microfactory/ (visited on 05/23/2016).

[178] Hartmut Weule and Carsten Buchholz. "Method for assessment of reuse suitability within modular assembly systems". In: *Assembly Automation* 21.3 (2001), pp. 241–246. ISSN: 0144-5154. DOI: 10.1108/01445150110398837.

[179] Robert K. Yin. *Case Study Research - Design an Methods*. SAGE Publications, Inc; 4th edition, 2009, p. 240. ISBN: 978-1-4129-6099-1. URL: http://www.amazon.com/Case-Study-Research-Methods-Applied/dp/1412960991.

[180] Chaim Zins. "Conceptual approaches for defining data, information, and knowledge". In: *Journal of the American Society for Information Science and Technology* 58.4 (2007), pp. 479–493. ISSN: 15322882. DOI: 10.1002/asi.20508.

Appendices

A Table of Relevant External Descriptions

Table A.1: Summary of aspects of most relevant description languages

Description	Purpose	Feasibility for Thesis	Relations	Std./ Org.	Refs
Application Integration Framework (AIF)	Industrial automation systems and integration - Open systems application integration framework.	Framework. Too generic and high level.		ISO	Ch. 3.4.2.2 (p.41)
Assembly Module Platform (AMP)	Process-oriented product design concept				
Smart Components (AML / SC)	Extension to AutomationML in the form of digital representations of real objects, called SmartComponents.				Ch. 3.4.1.1 (p.39)
Automation Markup Language (AML)	Engineering data exchange format for use in industrial automation systems engineering. AML is an XML schema based data format and has been developed in order to support the data exchange in a heterogeneous engineering tools landscape. The goal of AML is to interconnect engineering tools in their different disciplines, e.g. mechanical plant engineering, electrical design, process engineering, process control engineering, HMI development, PLC programming, robot programming, etc.	System level description	Process Control Engineering (PCE)	IEC/ AutomationML e.V.	Ch. 3.4.1.1 (p.37) [4]
Computer Aided Engineering eXchange (CAEX)	Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools			IEC	Ch. 3.4.1.1 (p.40)
COBASA	Multiagent architecture based on coalitions of assembly modules (CoBASA)		Jose Barata PhD		
COLLAborative Design Activity (COLLADA)	COLLAborative Design Activity for establishing an interchange file format for interactive 3D applications. E.g. used as Robot description language. Defines XML-based schema to make it easy to transport 3D assets between applications.			/Khronos	

Continued on next page

Table A.1 – *Continued from previous page*

Description	Purpose	Feasibility for Thesis	Relations	Std./ Org.	Refs
Electronic Device Description (EDD)	Aim is to define process automation devices and to enable easy integration of these devices into a control system. Any device from very simple sensor or valve to a complex one like drive. The description supplies information of all device parameters, for GUI, maintenance, and diagnostics functions. The focus of description is on the area of process automation.	Focuses on process automation domain and supports only few fieldbus systems in process automation (HART, Foundation Fieldbus, and Profibus).			Ch. 3.4.2.3 (p.44)
Electronic Device Description Language (EDDL)	Device description at process automation. Contains information about all device paramters, GUI, maintenance, diagnostics, and datasheet information of the device.	Focuses on process automation domain. ASCII + C -language based and misses (hierarchical) structuring. Applicable mostly in commissioning and operations phases. Support for selection and design is low.	EDD		Ch. 3.4.2.3 (p.44)
Field Device Configuration Markup Language (FDCML)	FDCML describes various perspectives of automation component. Some examples are the identification, communication skills, functions, diagnostic information, and mechanical description. This allows different applications to evaluate different aspects of a component. FDCML can describe automation devices from simple interface converters to gateways with several protocol stacks. Included are : Definition of profiles as a source of device development; Source for device data sheets; Electrical Description; and Electrical Device description as input for engineering tools			ISO/ Interbus,Phoenix Contact	Ch. 3.4.2.3 (p.43)
Field Device Integration (FDI)	With FDI, a technology has been developed that combines the advantages of FDT with those of EDDL in a single solution. FDI takes account of the various tasks over the entire lifecycle for both simple and the most complex devices, including configuration, commissioning, diagnosis and calibration.		EDDL, FDT	IEC/ FDI Cooperation	Ch. 3.4.2.3 (p.48)

Continued on next page

Table A.1 – *Continued from previous page*

Description	Purpose	Feasibility for Thesis	Relations	Std./ Org.	Refs
Field Device Technology (FDT)	FDT standardises the communication and configuration interface between all field devices (sensors, actuators) and host systems over number of fieldbusses. Framework for GUI integration. Includes a device-specific software component, DTM (Device Type Manager), is supplied by the field device manufacturer with its device. The DTM is integrated into engineering tools via the FDT interfaces.	FTD focus mainly on process automation. It represents process automation field devices at fieldbus level, which is too low level. Bases on Microsoft .NET technology.	EDD / DD, IEC 61784	IEC/ FDT Group AISBL	Ch. 3.4.2.3 (p.46)
MANDATE	Industrial automation systems and integration – Industrial manufacturing management data: Resources usage management – Part 32: Conceptual model for resources usage management data. It is a product-process-resource based approach for managing modularity in production management.			ISO	Ch. 3.4.1.3 (p.40)
Module Process Description (MPD)	Defines the processes for assembly modules		AMP		
Manufacturing Service Description (MSD)	Manufactruring-as-a-Service implementation. Manufacturing network description. What to produce and how MaaS network can be utilised for producing the required product. Manuf Service Desc = what resources and with what capabilities exists in the virtual production network.				
Open Electronic Device Description Markup Language (Open-EDDML)	XML formatted version of EDDL. That is a device description for process automation. It contains information about all device paramters, GUI, maintenance, diagnostics, and datasheet information of the device.		EDD, EDDL		Ch. 3.4.2.3 (p.45)
Web Ontology Language (OWL)	OWL is a family of knowledge representation languages for authoring ontologies. The languages are characterised by formal semantics and RDF/XML-based serializations for the Semantic Web. OWL is endorsed by the W3C and has attracted academic, medical and commercial interest.	Too wide and complex. Could be applied by expressing process taxonomies/ontologies?		W3C	
PLCopen <XML>	Define PLC logic and operations in XML format				Ch. 3.4.4 (p.49)

Continued on next page

Table A.1 – *Continued from previous page*

Description	Purpose	Feasibility for Thesis	Relations	Std./ Org.	Refs
Process Specification Language (PSL)	Defines common ontology between N supply chain partners. The aim is to provide a middle-level exchange layer, so that every connecting partner needs to implement only one single, common interface to communicate with others.			ISO	Ch. 3.4.3 (p.48)
Resource Description Framework (RDF)	Resource Description Framework (RDF) is a family of W3C specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax formats.			W3C	
Service Modelling Language (SML)	SML define a set of XML instance document extensions for expressing links between elements, a set of XML Schema extensions for constraining those links, and a way to associate Schematron rules with global element declarations, global complex type definitions, and/or model documents. Area: information technology				Ch. 3.4.4 (p.50)
Service oriented architecture Modeling Language (SoaML)	SoaML is an open source specification, describing a UML profile and metamodel for the modeling and design of services within a service-oriented architecture.		UML	OMG	Ch. 3.4.4 (p.50)
Systems Modelling Language (SysML)	SysML is a requirement driven, general purpose modelling language used for system engineering. It is a general-purpose graphical modelling language for specifying, analysing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modelling system requirements, behaviour, structure, and parametrics, which is used to integrate with other engineering analysis models.		UML	OMG/ OMG	Ch. 3.4.1.2 (p.40)

Continued on next page

Table A.1 – *Continued from previous page*

Description	Purpose	Feasibility for Thesis	Relations	Std./ Org.	Refs
Unified Manufacturing Resource Model (UMRM)	UMRM is a modelling method for CNC machine tools and their auxiliary devices like tools, tool holders, fixtures, pallets, conveyors, handling robot, etc. It is an unified model to exchange machine tool related resource information between CAPP tools. It utilises STEP and STEP-NC, and represents information in compatible form.	Focuses only on the domain of machine tools and models contains strong typing. Its generalisation to other domains remains questionable.	STEP		Ch. 3.4.2.1 (p.41)
Unified Robot Description Format (URDF)	Describes the physical properties of the robot in ROS, such as the lengths of the links, the sizes of its wheels and body, the locations of its sensors, and the visual appearance of each part of the robot. It contains the kinematics, inertia, visualisation, and simplified collision modelling-related information for the links.	Only 3D models and kinematics. Could be referenced and reutilised as part of description.	ROS	/ROS/ROS-industrial	Ch. 3.4.5.1 (p.51)
X3D	X3D is the ISO standard XML-based file format for representing 3D computer graphics, the successor to the Virtual Reality Modeling Language (VRML). X3D features extensions to VRML (e.g. Humanoid animation, NURBS, GeoVRML etc.), the ability to encode the scene using an XML syntax as well as the Open Inventor-like syntax of VRML97, or binary formatting, and enhanced application programming interfaces (APIs).	Only 3D models and kinematics. Could be referenced and reutilised as part of description.	VRML	ISO/IEC	
Electronic Device Description (XDD)	Since end of 2007 a new XML based format called XDD is defined in CiA311. XDD is conformant to ISO Standard 15745. For both formats a free editor is available, called CANeds.	Control and fieldbus orientation. Low level.		CANOpen	

B Listing of User Requirements for the Emplacement Concept

Not all of the presented User Requirements (URs) are applied directly to module descriptions. These are collected into Ch. 4.2.4.7 (p.68). There are requirements to systems utilising the descriptions, such as production system design tool, and posing an indirect requirements for the descriptions.

Generic requirements

UR 1. The concept must enable and contain mechanism(s) for exchangeability and interchangeability of production modules.

UR 2. Interoperability and compatibility between production modules is improved comparing the current situation.

UR 3. The concept and descriptions should promote and enable multi vendor production systems. System design is not limited to single supplier.

UR 4. Enhance openness and neutrality of the architecture, platform, Operating System (OS), programming environment, or other Software (SW) tools

UR 5. Open architecture is supported i.e accepting different architectures and not limiting to one specific architecture. Future architectures should be acceptable without modifications.

UR 6. Enhance modularity and the re-use of Hardware (HW) and SW components and description levels

UR 7. The descriptions are independent, autonomous, and self-contained. They can be used without resort to additional external resources.

UR 8. The descriptions are exact, unambiguous and clear.

UR 9. The concept and the descriptions are easy and simple to use

UR 10. Descriptions allow internationalisation

UR 11. Data integrity, security, and Intellectual Property (IP) protection measures are provided.
- a) Data corruption or tampering of descriptions can be recognised
- b) Concept and descriptions are secure to use
- c) Concept supports IP protection for production module
- d) Technical implementations are not revealed through the descriptions.

UR 12. The concept and the descriptions shall not inhibit technological development and the appearance of new processes, interfaces, and features.

UR 13. The module description needs to be:
- a) capable of representing the properties of real-world production modules,
- b) readable and writeable by both machine and human experts,
- c) comprehensive and complete,
- d) easy to make and maintain,
- e) unambiguous and exact.

UR 14. Include definitions for
- a) interface (how modules can be connected together), and
- b) capability (what a module can do)

UR 15. Include and contain module (design) parameters important for
- a) system design,
- b) system simulation and verification, and
- c) operation and execution.

UR 16. The description can be carried physically with the module and simultaneously be available on Internet.

UR 17. Use of International System of Units (SI).

UR 18. Power and fit of expression. The concept and associated descriptions need to be powerful and fitting for expressing the information needed for the different use-cases for design, deployment, and operation of the production system.
UR 19. The description language for the concept should contain optimised versions (layers) for serving different use-cases from the main stakeholders (users). However, the number of different layers should be minimised and they should contain as many similar data structures as possible.
UR 20. The descriptions can be verified
  a) by structure,
  b) to follow data models and their rules for restrictions and limitations, and
  c) by content (if possible)
UR 21. Production module description can be verified against templates. I.e. to verify that the module description is conducted correctly and respects the corresponding template.
UR 22. Verification should
  a) find errors and conflicting conditions and brought up, highlight, and/or report them
  b) be automatised process

Module Provider specific requirements

UR 23. Information like parameters must be adjustable and extendable later as all information (set of parameters as well their values) is not known prior.
UR 24. Legacy equipment should be possible to be presented with the concept / language.
UR 25. The descriptions should not reveal the IP of the *Module Provider*, but should support easy adoption and usage of the modules.
UR 26. The concept provides a marketing channel for the *Module Providers* to promote their offerings.
UR 27. The concept and descriptions should leave space for competition and differentiation.

System Designer specific requirements

UR 28. Production System designs are created and verified faster and more efficient than in present.
UR 29. Any comparison of the modules should be easier and quicker than it is at present.
UR 30. Selection of the modules should be quicker and more efficient.
UR 31. Easy access to the information about large set of modules from many module providers. Existing and/or available modules are easy to find.
UR 32. The concept and descriptions must eliminate, or at least reduce, the need to re-create, re-type, and search for information
UR 33. Descriptions information is formalised.
UR 34. Semantics of information is explained or offered.
UR 35. The module description needs to be:
  a) comprehensive and complete. I.e. it should include all aspects *System Designer* needs while making the production system design and component selection and verification.
UR 36. The system design should have clear structure (easier to understand, design, and construct).
UR 37. The concept and descriptions should support an iterative system design concepts, which can be implied by any of the three alternative approaches or any combination of these:
  a) Top down,
  b) middle out, or

c) bottom up.

UR 38. The information available in the descriptions shall enable implementation of automatised selection and verification operations for system designer. These automatised operations include procedures like[1]:
   a) Searching and limiting the offerings to the fitting modules according the selected criteria such as architecture, interface, skill or capability, function, feature, and value or value range for property.
   b) Verify the feasibility of the system design (fitness, completeness).
   c) Analysis of the system design in theoretical fashion (performance, throughput, tack time, quality, bottlenecks, etc.).
   d) Verification of existing system design for a new production scenario and tracing the missing process sequences or mismatches in process parameters.
   e) Evaluation of work envelopes and check for collisions. First level would be static analysis (will all stationary modules fit together) and secondly dynamic case. The later would include program verification for possible collisions.
   f) The system assembly could be analysed. I.e. assembly order, is there enough clearance to place components together, etc.
   g) Automatised creation of system designs from predefined set of modules.

System Integrator specific requirements

UR 39. Building blocks shall be self contained and independent of external information sources.

UR 40. Modules can be independently tested and calibrated. (Also UR from End User/Maintenance)
   a) The concept and descriptions help on defining and implementing unit tests and calibrations.
   b) *Module providers* should deliver (unit) tested modules.
   c) Modules should have self test or build-in test procedures implemented.
   d) Unit test can be performed later e.g. in case of maintenance operations.

UR 41. Integration and system set-up times should be minimised and these must be definitely shorter than in present.

UR 42. System shall have clear structure (easier to understand, design, and construct)

UR 43. Minimise need for custom designed or customised modules. Survive as far as possible with standard Commercially-available Off-The-Shelf (COTS) modules.

UR 44. Communication and connection capabilities between modules and to next Computer Integrated Manufacturing (CIM) hierarchical levels.
   a) Units composing a cell can conclude and report its state. The cell or module is capable to report is status and operations to Manufacturing Execution System (MES) and Enterprise Resource Planning (ERP) systems.
   b) Communication links are set and established easily.

UR 45. Descriptions should support generating and configuring the controls instead of programming the entire system. This includes logic operations and Graphical User Interface (GUI).
   a) Modules contain their own code and processing capacity (I.e. they are self contained). Configurations are made via connecting the module interfaces from a point to another, passing parameters to inputs, etc.
   b) Control connections between modules are fast and easy to make and change. They can be even generated automatically.

[1] The implementation of the procedures and tools required for them are exclude from the scope of this thesis.

c) GUIs can be generated on-the-fly from specifications included into production module description.

End User specific requirements

UR 46. The concept should improve production system response i.e. reduce TTM and TTV.
UR 47. The concept is prepared for change of
  a) manufactured products and product variants,
  b) production volume (increase/decrease).
UR 48. The concept should improve re-use, resale, and utility value of production modules after decommission.

End User specific requirements (System operator):

UR 49. The production system should be more visible, accessible, and controllable.
UR 50. Individual modules and system are easier to operate.
UR 51. Similar look and feel (automatically generated GUIs).
UR 52. Similar operating principles of production modules.

End User specific requirements (Maintenance staff):

UR 53. Easier to maintain (Accessible, detachable modules, ...).
UR 54. Minimise or reduce downtime after resource breakdown.
UR 55. Change of module due course of breakdown, upgrade, or other reasons, to a same kind of, similar, or even another from different supplier, is possible with minimum intervention to other parts of production system.

Module Owner specific requirements

UR 56. Create concept and methods to keep modules up-to-date and eligible $\rightarrow$ securing the investment (enable longer redemption times, and improve resale and utility values).
UR 57. Enable and provide means for efficient resource utilisation.
UR 58. Improve sustainability and re-use compared to present.
UR 59. Offer the means to find available production capability and capacity quicker and easier.

Harmonisation Organisation specific requirements

UR 60. Offer the means (tools) to process descriptions, i.e. to create, maintain, and modify them.
UR 61. Offer the means (tools) to distribute descriptions, i.e. to manage and share them.
UR 62. Have means to monitor the templates and descriptions with features like version control and trace of changes.
UR 63. All other users shall have possibility to affect the templates and central definitions. However, the whole process should be need based, open, and unbiased between the parties.
UR 64. The process for development, review, change of descriptions need to be defined and documented.
UR 65. The objective is to limit the amount of templates to minimum.
UR 66. To have the means (tools) to identify and resolve conflicting template proposals.

C Model of the Emplacement

C.1 Emplacement File

The code listing below shows an excerpt of the first 58 lines of a Emplacement file (Specifically GripperEmplacement.xml). This is for getting a feeling what kind of entity this is and what is defined in here. The complete file can be find from `http://resourcedescription.tut.fi/EmplacementWS/emplacements/GripperEmplacement.xml`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- edited with XMLSpy v2007 (http://www.altova.com) by Niko Siltala (TUT /
       IPE) -->
3  <?xml-stylesheet type="text/xsl" href="http://emplacementws.tte.tut.fi/
       EmplacementWS/stylesheets/Empl_FormatterEmplDoc.xsl"?><!--Alternative
       schema location for the local and internet usage
4  ../schemas/EUPASSEmplacement.xsd
5  http://emplacementws.tte.tut.fi/EmplacementWS/schemas/EUPASSEmplacement.xsd
6  -->
7  <Emplacement xmlns="http://www.eupass.org/2005/XMLSchema" xmlns:xsi="http://www
       .w3.org/2001/XMLSchema-instance" xmlns:ns_1="http://www.w3.org/2001/
       XMLSchema-instance" ns_1:schemaLocation="http://www.eupass.org/2005/
       XMLSchema http://emplacementws.tte.tut.fi/EmplacementWS/schemas/
       EUPASSEmplacement_v1-0-0.xsd" emplacementID="gripper.1" emplacementName="
       Emplacement - gripper.1" created="2007-01-01T00:00:00+03:00" authors="Niko
       Siltala, Andreas Hofmann, Herbert Vollmer, EUPASS/AT/MWG, EUPASS/AT/WG4"
       modified="2008-06-04T13:00:00+02:00" version="0.7" emplLevel="DEVICE"
       schemaVersion="1.0.0">
8    <OverviewAndScope>
9      <Introduction>This Emplacement specifies grippers. Grippers are the modules
        most often interacting with the handled product at the end of some kind of
        manipulator.
10     </Introduction>
11     <Scope>All kind of grippers shall be included taking no stand of the
       gripping technology used.</Scope>
12   </OverviewAndScope>
13   <GeneralRequirements>
14     <XHTMLDescription>TBD General requirements....</XHTMLDescription>
15     <References>
16       <Reference id="ref.ISO_8373" name="ISO 8373">
17         <XHTMLDescription>Manipulating industrial robots - Vocabulary.</
       XHTMLDescription>
18       </Reference>
19       <Reference id="ref.ISO_9787" name="ISO 9787">
20         <XHTMLDescription>Manipulating industrial robots - Coordinate systems
       and motion nomenclatures.<br/>This International Standard defines and
       specifies robot coordinate systems. It also provides a nomenclature
       including notation for the basic robot motions. It is intended to aid in
       robot alignment, testing, and programming.</XHTMLDescription>
21       </Reference>
22       <Reference id="ref.ISO_14539" name="ISO 14539">
23         <XHTMLDescription>Manipulating industrial robots - Object handling with
        grasp-type grippers - Vocabulary and presentation of characteristics<br/>
       This International Standard provides terms to describe object handling and
       terms of functions, structures, and elements of grasp-type grippers.</
       XHTMLDescription>
24       </Reference>
25     </References>
26     <Terms>
27       <TermDefinition id="term.TCP" term="TCP / Tool Centre Point">
28         <XHTMLDescription>Point defined for a given application with regard to
       the mechanical interface coordinate system (as defined in ISO 8373) <br/>
```

```
    NOTE 1 The TCP is the origin of the tool coordinate system (as defined in
    ISO 9787).
<br/>NOTE 2 The TCP can be considered as an important point of agreement
    between manufacturers and users for handling objects and for each end
    effector. [ISO 14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.grasping" term="grasping">
        <XHTMLDescription>Gripper's motion to apply constraints by finger(s) to
     an object. [ISO 14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.releasing" term="releasing">
        <XHTMLDescription>Gripper's motion to eliminate constraints from an
    object. [ISO 14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.grasp" term="grasp">
        <XHTMLDescription>Constraints of an object with gripper finger(s). [ISO
     14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.grip" term="grip">
        <XHTMLDescription>Constraints of an object by an end effector. [ISO
    14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.handling.object" term="object handling">
        <XHTMLDescription>Action on an object by an end effector, or keeping a
    state of an object by an end effector. [ISO 14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.grasp.external" term="external/outside grasp">
        <XHTMLDescription>Grasp that effects on the external surface of the
    object. [ISO 14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.grasp.internal" term="internal/inside grasp">
        <XHTMLDescription>Grasp that effects on the internal surface of the
    object. [ISO 14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.grasp.closure.form" term="form closure grasp">
        <XHTMLDescription>Grasp with degrees of freedom of object being 0 or
    less without considering friction forces at contact points.<br/>NOTE Form
    closure grasp is a grasp in which only the configuration of the gripper
    defines the pose of the object. [ISO 14539]</XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.grasp.closure.force" term="force closure grasp">
        <XHTMLDescription>Grasp with degrees of freedom of object being 1 or
    more without considering friction forces at contact points but 0 or less
    with considering them. <br/>NOTE Force closure grasp is a grasp in which
    not only the configuration of the gripper but also the forces serve to keep
     the pose of the object. Forces are usually friction forces. [ISO 14539]</
    XHTMLDescription>
      </TermDefinition>
      <TermDefinition id="term.TCS" term="tool coordinate system">
```

C.2 Emplacement Schema

The code listing below shows an excerpt of the first 50 lines of a Emplacement XML Schema Definition (XSD) file. This is for getting a feeling what kind of entity this is and what is defined in here. The complete file can be found from `http://resourcedescription.tut.fi/EmplacementWS/schemas/EUPASSEmplacement_v1-0-0.xsd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by Niko Siltala (
    Tampere University of Technology) -->
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:me="http://www.eupass.org/2005/XMLSchema" targetNamespace="http://www.eupass.org/2005/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
  <xs:include schemaLocation="./EUPASSBaseTypes_v1-0-0.xsd"/>
  <xs:element name="Emplacement">
    <xs:annotation>
      <xs:documentation>ROOT of emplacement META language</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="OverviewAndScope" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Overview and scope definition of this emplacement</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Introduction" type="me:XHTML_Doc_CT"/>
              <xs:element name="Scope" type="me:XHTML_Doc_CT"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="GeneralRequirements" minOccurs="0">
          <xs:annotation>
            <xs:documentation>General requirements for this emplacement</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="me:XHTMLDescription" minOccurs="0"/>
              <xs:element name="References">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element ref="me:Reference" minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="Terms">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element ref="me:TermDefinition" minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="TheoryOfOperation" type="me:XHTML_Doc_CT" minOccurs="0">
          <xs:annotation>
            <xs:documentation>How this emplacement shall function and operate</xs:documentation>
          </xs:annotation>
        </xs:element>
```

D Model of the Blueprint

D.1 Blueprint File

The code listing below shows an excerpt of the first 46 lines of a BP file (BP_Festo_PV2_Gripper1.xml). This is for getting a feeling what kind of entity this is and what is defined in here. The complete file can be found from `http://resourcedescription.tut.fi/EmplacementWS/bluePrints/BP_Festo_PV2_Gripper1.xml`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="../stylesheets/BP_FormatterBPDoc.xsl"?>
3  <!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by Niko Siltala (
       Tampere University of Technology) -->
4  <ModuleDescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
       xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.eupass.org/2005/
       XMLSchema" xmlns:ns_1="http://www.w3.org/2001/XMLSchema-instance" ns_1:
       schemaLocation="http://www.eupass.org/2005/XMLSchema http://emplacementws.
       tte.tut.fi/EmplacementWS/schemas/EUPASSBluePrint_v1-0-0.xsd" schemaVersion=
       "1.0.0" version="" author="" email="" created="2008-06-30T18
       :49:56.5861295+03:00">
5    <Module id="Festo_Gripper_no1_for_EUPASS_PV2" category="PROCESS_MODULE">
6      <Info>
7        <Label>
8          <String lang="en">Festo_Gripper_no1_for_EUPASS_PV2</String>
9        </Label>
10       <Comment>
11         <String lang="en">Festo Gripper No1. for EUPASS PV2. - Two finger
       tactile gripper with force control.</String>
12       </Comment>
13       <Documentation>
14         <Link>
15           <Label lang="en">TODO_OPT_DocumentationLabel</Label>
16           <URI>http://www.festo.com</URI>
17         </Link>
18       </Documentation>
19       <PhysicalProperties>
20         <Mass value="0.229" unit="kg"/>
21         <InstallationPositions limited="false"/>
22         <CentreOfGravity x="-0.00323" y="-0.003" z="0.03"/>
23       </PhysicalProperties>
24       <EnvironmentalProperties>
25         <EnvironmentalProperty name="Temperature during operation" description=
       "Acceptable temperature range during the Operation" increment="0.1" unit="
       deg" decade="0" id="propEnv.op.temp" datatype="LREAL" reqOpt="REQ" min="5"
       max="40" value="25" classOfProperty="NORMAL_OP"/>
26         <EnvironmentalProperty name="Module as temperature source during
       operation" description="Temperature range generated during the Operation.
       Module as source of heat." increment="0.1" unit="deg" decade="0" id="
       propEnv.op.temp.source" datatype="LREAL" reqOpt="REQ" min="0" max="40"
       value="25" classOfProperty="NORMAL_OP"/>
27         <EnvironmentalProperty name="Temperature during transportation"
       description="Acceptable temperature range during the Trasport or Storage"
       increment="0.1" unit="deg" decade="0" id="propEnv.trans.temp" datatype="
       LREAL" reqOpt="REQ" min="5" max="40" value="25" classOfProperty="TRANSPORT"
       />
28         <EnvironmentalProperty name="Humidity during operation" unit="%" decade
       ="0" id="propEnv.op.humidity" datatype="LREAL" reqOpt="OPT" min="0" max="60
       " classOfProperty="NORMAL_OP"/>
29         <EnvironmentalProperty name="Shock during operation" description="
       Acceptable shock range for the module during the operation" unit="g" decade
       ="0" id="propEnv.op.shock" datatype="LREAL" reqOpt="OPT" min="0" max="3"
       classOfProperty="NORMAL_OP"/>
```

```
30        <EnvironmentalProperty name="Humidity during service" unit="%" decade="
      0" id="propEnv.service.humidity" datatype="LREAL" reqOpt="OPT" min="0" max=
      "60" classOfProperty="SERVICE"/>
31        <EnvironmentalProperty name="Shock during service" description="
      Acceptable shock range for the module during the service" unit="g" decade="
      0" id="propEnv.service.shock" datatype="LREAL" reqOpt="OPT" min="0" max="30
      " classOfProperty="SERVICE"/>
32        <EnvironmentalProperty name="Temperature during service" description="
      Acceptable temperature range during the Service of the module." increment="
      0.1" unit="deg" decade="0" id="propEnv.service.temp" datatype="LREAL"
      reqOpt="OPT" min="5" max="40" value="25" classOfProperty="SERVICE"/>
33        <EnvironmentalProperty name="Humidity during transportation" unit="%"
      decade="0" id="propEnv.trans.humidity" datatype="LREAL" reqOpt="OPT" min="0
      " max="60" classOfProperty="TRANSPORT"/>
34        <EnvironmentalProperty name="Shock during transportation" description="
      Acceptable shock range for the module during the transportation" unit="g"
      decade="0" id="propEnv.trans.shock" datatype="LREAL" reqOpt="OPT" min="0"
      max="30" classOfProperty="TRANSPORT"/>
35      </EnvironmentalProperties>
36      <BusinessProperties>
37        <BusinessProperty id="propBus.calibration.interval" name="Average
      calibration interval" description="Average calibration interval" unit="h"
      decade="0" datatype="TIME" reqOpt="REQ" min="0" max="0" value="0"
      classOfProperty="CONFIGURATION"/>
38        <BusinessProperty id="propBus.calibration.time" name="Calibration time"
       description="Time required for calibrating an adjusted offset of the
      module" unit="h" decade="0" datatype="TIME" reqOpt="REQ" min="0" max="0"
      value="0" classOfProperty="CONFIGURATION"/>
39        <BusinessProperty id="propBus.failure.recoverytime.avg" name="Failure
      Recovery Time" description="Average failure recovery time" unit="h" decade=
      "0" datatype="TIME" reqOpt="REQ" min="0.01667" max="0" value="0.06667"
      classOfProperty="MAINTENANCE"/>
40        <BusinessProperty id="propBus.installation.time" name="Installation
      time" description="Time required for installing module" unit="h" decade="0"
       datatype="TIME" reqOpt="REQ" min="0" max="0" value="0" classOfProperty="
      CONFIGURATION"/>
41        <BusinessProperty id="propBus.maintenance.frequency" name="Maintenance
      Frequency" description="Average maintenance frequency" unit="1/month"
      decade="0" datatype="LREAL" reqOpt="REQ" min="0" max="0" value="0"
      classOfProperty="MAINTENANCE"/>
42        <BusinessProperty id="propBus.maintenance.time" name="Maintenance Time"
       description="Average maintenance time" unit="h" decade="0" datatype="TIME"
       reqOpt="REQ" min="0" max="0" value="0" classOfProperty="MAINTENANCE"/>
43        <BusinessProperty id="propBus.module.cost.consumable" name="Consumable
      Cost" description="Cost of material consumed by the module (glue, oil,
      energy, etc.)" unit="EUR/month" decade="0" datatype="LREAL" reqOpt="REQ"
      min="0" max="0" value="10" classOfProperty="INVESTMENT"/>
44        <BusinessProperty id="propBus.module.cost.purchase" name="Purchase Cost
      " description="Purchasing cost of the module. The list price." unit="EUR"
      decade="0" datatype="LREAL" reqOpt="REQ" min="0" max="0" value="100"
      classOfProperty="INVESTMENT"/>
45        <BusinessProperty id="propBus.module.cost.rental" name="Rental Cost"
      description="Renting cost of the module." unit="EUR/month" decade="0"
      datatype="LREAL" reqOpt="REQ" min="0" max="0" value="100" classOfProperty="
      INVESTMENT"/>
46        <BusinessProperty name="Average lifetime of the module in time"
      description="Average lifetime of the module. Effective hours." unit="h"
      decade="0" id="propBus.module.lifetime.time" datatype="TIME" reqOpt="REQ"
      min="0" classOfProperty="GENERIC"/>
```

D.2 Blueprint Schema

The code listing below shows an excerpt of the first 44 lines of a BP XSD file. This is for getting a feeling what kind of entity this is and what is defined in here. The complete file can be found from `http://resourcedescription.tut.fi/EmplacementWS/schemas/EUPASSBluePrint_v1-0-0.xsd`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by Niko Siltala (
       Tampere University of Technology) -->
3  <xs:schema xmlns:eupass="http://www.eupass.org/2005/XMLSchema" xmlns:xs="http
       ://www.w3.org/2001/XMLSchema" targetNamespace="http://www.eupass.org/2005/
       XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified
       " version="0.9">
4    <xs:include schemaLocation="./EUPASSBaseTypes_v1-0-0.xsd"/>
5    <xs:element name="ModuleDescription">
6      <xs:annotation>
7        <xs:documentation>Module Description. The MAIN ROOT OF THE DESCRIPTION</
       xs:documentation>
8      </xs:annotation>
9      <xs:complexType>
10       <xs:sequence>
11         <xs:element name="Module" type="eupass:Module_CT" maxOccurs="unbounded"
       >
12           <xs:annotation>
13             <xs:documentation>The Blueprint can contain more than one "virtual"
        Module that can be handled externally and described as seperate Modules.
       However the delivery will be preassembled in one piece.</xs:documentation>
14           </xs:annotation>
15         </xs:element>
16         <xs:element name="DescriptionVersion" type="eupass:Version_CT">
17           <xs:annotation>
18             <xs:documentation>Version of the Description file</xs:documentation
       >
19           </xs:annotation>
20         </xs:element>
21         <xs:element name="SerialNumber" type="xs:positiveInteger">
22           <xs:annotation>
23             <xs:documentation>Serial Number of the Device </xs:documentation>
24           </xs:annotation>
25         </xs:element>
26         <xs:element name="Extension" type="eupass:ExtensionData_CT" minOccurs="
       0" maxOccurs="unbounded">
27           <xs:annotation>
28             <xs:documentation>Custom Extensions of the Module Description</xs:
       documentation>
29           </xs:annotation>
30         </xs:element>
31       </xs:sequence>
32       <xs:attribute name="schemaVersion" type="xs:string" use="required"/>
33       <xs:attribute name="version" type="xs:string" use="required"/>
34       <xs:attribute name="author" type="xs:string" use="required"/>
35       <xs:attribute name="email" type="xs:string" use="required"/>
36       <xs:attribute name="company" type="xs:string" use="optional"/>
37       <xs:attribute name="phone" type="xs:string" use="optional"/>
38       <xs:attribute name="created" type="xs:dateTime" use="required"/>
39       <xs:attribute name="modified" type="xs:dateTime" use="optional"/>
40     </xs:complexType>
41   </xs:element>
42   <xs:complexType name="Module_CT">
43     <xs:annotation>
44       <xs:documentation>Eupass module type</xs:documentation>
```

E Usage of eXtensible Stylesheet Language Transformation (XSLT)

The following sections represent some examples of eXtensible Stylesheet Language Transformations (XSLTs) used within the implementation. These can be used to transform the eXtensible Markup Language (XML) content to another formats.

E.1 XSLT for Emplacement Documentation

The code listing below shows an excerpt of the first 30 lines and lines 89 to 162 of a XSLT file for generating the human readable document out of given Emplacement file. This is for getting a feeling what kind of entity this is and what is defined in here. The complete file can be found from `http://resourcedescription.tut.fi/EmplacementWS/stylesheets/Empl_FormatterEmplDoc.xsl`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:me="http://www.eupass.
       org/2005/XMLSchema" xmlns:xpath="http://www.w3.org/2005/02/xpath-functions"
        xmlns="http://www.w3.org/1999/xhtml">
3    <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
4    <!--iso-8859-1
5    Removed XHTML definition from output
6      doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN" doctype-system="http://
       www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
7    -->
8    <!--Creartion of XSL Keys-->
9    <xsl:key name="keyProfile" match="me:Profile" use="@id"/>
10   <xsl:key name="keyAction" match="me:Action" use="@id"/>
11   <xsl:key name="keyInterface" match="me:Interface" use="@id"/>
12   <xsl:key name="keyVariable" match="me:Variable" use="@id"/>
13   <xsl:key name="keyVariableB" match="me:VariableBoolean" use="@id"/>
14   <xsl:key name="keyVariableN" match="me:VariableNum" use="@id"/>
15   <xsl:key name="keyVariableS" match="me:VariableString" use="@id"/>
16   <xsl:key name="keyBusinessProperty" match="me:BusinessProperty" use="@id"/>
17   <!--XSL Variables-->
18   <xsl:variable name="color.table.header">#99CCFF</xsl:variable>
19   <xsl:variable name="table.width.variable">13</xsl:variable>
20   <xsl:variable name="URLBase">http://emplacementws.tte.tut.fi/EmplacementWS</
       xsl:variable>
21   <xsl:variable name="headerInterfaceRef">
22     <tr bgcolor="{$color.table.header}">
23       <th align="left">EmplacementIFID</th>
24       <th align="left">Description</th>
25       <th align="left">Interface Ref</th>
26       <th align="left">Gender</th>
27       <th align="left">Req/Opt</th>
28       <th align="left">Occ.</th>
29     </tr>
30   </xsl:variable>
```

```
89   <!--START OF MAIN HTML DOCUMENT BODY
       ************************************************************************-->
90   <!--
91   Processing the document instance
92   -->
93   <xsl:template match="/*">
94     <html>
95       <!--Insert Comment Note into the generated outputfile-->
96       <xsl:text disable-output-escaping="yes">&lt;!--NOTE! CAUTION! THIS FILE
       IS AUTOMATICALLY GENERATED!!!! DO NOT EDIT MANUALLY!!!--&gt;</xsl:text>
```

```
97        <head>
98          <!--<xsl:element name="meta">
99            <xsl:attribute name="http-equiv">Content-Type</xsl:attribute>
100           <xsl:attribute name="content">text/html; charset=iso-8859-1</xsl:
      attribute>
101       </xsl:element>-->
102       <title>Definition of emplacement: <xsl:value-of select="//me:
      Emplacement/@emplacementName"/>
103       </title>
104       <xsl:element name="link">
105         <xsl:attribute name="rel">stylesheet</xsl:attribute>
106         <xsl:attribute name="type">text/css</xsl:attribute>
107         <xsl:attribute name="href"><xsl:value-of select="$URLBase"/>/
      stylesheets/Empl_StyleSheet_emplacement.css</xsl:attribute>
108       </xsl:element>
109       <!--:text disable-output-escaping="yes"><link rel="stylesheet" type="
      text/css" href="./schemas/StyleSheet_emplacement.css">
110       </xsl:text>-->
111     </head>
112     <body>
113       <p>
114         <h1>This is specification of emplacement: <xsl:value-of select="//me:
      Emplacement/@emplacementName"/>
115         </h1>
116       </p>
117       <table border="0">
118         <tr>
119           <td align="right">Name:</td>
120           <td>
121             <xsl:value-of select="//me:Emplacement/@emplacementName"/>
122           </td>
123         </tr>
124         <tr>
125           <td align="right">ID:</td>
126           <td>
127             <xsl:value-of select="//me:Emplacement/@emplacementID"/>
128           </td>
129         </tr>
130         <tr>
131           <td align="right">Emplacement Level:</td>
132           <td>
133             <xsl:value-of select="//me:Emplacement/@emplLevel"/>
134           </td>
135         </tr>
136         <tr>
137           <td align="right">Version:</td>
138           <td>
139             <xsl:value-of select="//me:Emplacement/@version"/>
140           </td>
141         </tr>
142         <tr>
143           <td align="right">Authors:</td>
144           <td>
145             <xsl:value-of select="//me:Emplacement/@authors"/>
146           </td>
147         </tr>
148         <tr>
149           <td align="right">Created / Modified:</td>
150           <td>
151             <xsl:value-of select="//me:Emplacement/@created"/> / <xsl:
      value-of select="//me:Emplacement/@modified"/>
152           </td>
```

```
153           </tr>
154           <!--<tr>
155               <td align="right">This doc created:</td>
156               <td><xsl:value-of select="current-dateTime()"/></td>
157             </tr>-->
158         </table>
159         <hr/>
160         <p>
161           <b>Table of Contents:</b>
162           <ol>
```

E.2 XSLT for BP Skeleton

The code listing below shows an excerpt of the first 131 lines of a XSLT file for generating the BP file skeleton out of given Emplacement file and Profile (Profile). This is for getting a feeling what kind of entity this is and what is defined in here. The complete file can be found from `http://resourcedescription.tut.fi/EmplacementWS/stylesheets/Empl_FormatterBPSkeleton.xsl`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- This XSL is created by Niko Siltala / Tampere University of Technology (
       TUT) / IPE
3       niko.siltala(at)tut.fi-->
4  <xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:me="http://www.eupass.org/2005/XMLSchema" xmlns="http://www.eupass.
       org/2005/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
       omit-xml-declaration="no" cdata-section-elements="namelist" media-type="
       text/xml"/>
6    <xsl:namespace-alias stylesheet-prefix="me" result-prefix="#default"/>
7    <!--<xsl:namespace-alias stylesheet-prefix="me2" result-prefix="#default"/>--
       >
8    <!--Creation of XSL Keys-->
9    <xsl:key name="keyProfile" match="me:Profile" use="@id"/>
10   <xsl:key name="keySkill" match="me:Skill" use="@id"/>
11   <xsl:key name="keyInterface" match="me:Interface" use="@id"/>
12   <!--XSL Variables-->
13   <xsl:variable name="var.profile">prof.gripper.2-finger_simple.1</xsl:variable
       >  <!--prof.Sample.real.ext.3-->
14   <xsl:variable name="URLBase">http://emplacementws.tte.tut.fi/EmplacementWS</
       xsl:variable>
15   <!--
16   Processing the document instance
17   -->
18   <xsl:template match="/">
19     <!--The processing instruction prevents user to see and save the
       description in XML format but instead in HTML
20       <xsl:processing-instruction name="xml-stylesheet">type="text/xsl" href="<
       xsl:value-of select="$URLBase"/>/stylesheets/BP_FormatterBPDoc.xsl"</xsl:
       processing-instruction>-->
21     <xsl:text disable-output-escaping="yes">
22     &lt;!--&lt;?xml-stylesheet type="text/xsl" href="</xsl:text>
23     <xsl:value-of select="$URLBase"/>
24     <xsl:text disable-output-escaping="yes">/stylesheets/BP_FormatterBPDoc.xsl
       "?&gt;--&gt;
25     </xsl:text>
26     <ModuleDescription version="0.1" author="TODO_Author(s)" email="TODO_name.
       name@comany.com" company="TODO_CompanyName" phone="TODO_Phone">
27       <xsl:attribute name="schemaVersion">1.0.0</xsl:attribute>
28       <xsl:attribute name="created">2014-07-11T15:30:00+03:00</xsl:attribute> <
       !--<xsl:value-of select="current-dateTime()"/>-->
```

```
29      <xsl:attribute name="modified">2014-07-11T15:30:00+03:00</xsl:attribute>
        <!--<xsl:value-of select="current-dateTime()"/>-->
30      <xsl:attribute name="xsi:schemaLocation">http://www.eupass.org/2005/
    XMLSchema http://emplacementws.tte.tut.fi/EmplacementWS/schemas/
    EUPASSBluePrint_v1-0-0.xsd</xsl:attribute>
31      <!--Insert Comment Note into the generated outputfile-->
32      <xsl:text disable-output-escaping="yes">
33      &lt;!--NOTE! CAUTION! THIS FILE IS AUTOMATICALLY GENERATED!
34      Intention is to offer a skeleton for Blue Print file editors!--&gt;
35      </xsl:text>
36      <Module id="TODO_ModuleID" category="SUPERVISOR">
37        <!--Create Info element-->
38        <Info>
39          <Label>
40            <String lang="en">TODO_ModuleLabel</String>
41          </Label>
42          <Comment>
43            <String lang="en">TODO_OPT_ModuleComment</String>
44          </Comment>
45          <Documentation>
46            <Link>
47              <Label lang="en">TODO_OPT_DocumentationLabel</Label>
48              <URI>http://www.TODO_company.com</URI>
49            </Link>
50          </Documentation>
51          <PhysicalProperties>
52            <Mass unit="kg" value="0"/>
53            <InstallationPositions limited="false"/>
54            <CentreOfGravity x="0.0" y="0.0" z="0.0"/>
55          </PhysicalProperties>
56          <EnvironmentalProperties>
57            <!--TODO make copying-->
58            <!--Picks the wanted profile-->
59            <xsl:apply-templates select="key('keyProfile', $var.profile)" mode=
    "withEnvironmentalProperties">
60              <xsl:sort select="@id" order="ascending"/>
61            </xsl:apply-templates>
62          </EnvironmentalProperties>
63          <BusinessProperties>
64            <!--TODO make copying-->
65            <!--Picks the wanted profile-->
66            <xsl:apply-templates select="key('keyProfile', $var.profile)" mode=
    "withBusinessProperties">
67              <xsl:sort select="@id" order="ascending"/>
68            </xsl:apply-templates>
69          </BusinessProperties>
70          <File_Image format="PNG" name="TODO_NameOfImage" URL="http://www.
    TODO_company.com">
71            <XHTMLDescription>TODO_OPT_XHTML_Description</XHTMLDescription>
72          </File_Image>
73          <File_CAD format="STEP" name="TODO_NameOfCADFile" URL="http://www.
    TODO_company.com">
74            <XHTMLDescription>TODO_OPT_XHTML_Description</XHTMLDescription>
75          </File_CAD>
76        </Info>
77        <!--Create Vendor element-->
78        <Vendor id="TODO_name">
79          <Name>
80            <String lang="en">TODO_nameOfCompany</String>
81          </Name>
82          <Link>
83            <Label lang="en">TODO_labelOfWebPage</Label>
```

```
84            <URI>http://www.TODO_company.com/</URI>
85          </Link>
86        </Vendor>
87        <!--Create Emplacement element-->
88        <Emplacement>
89          <xsl:attribute name="emplacementIDRef"><xsl:value-of select="/me:
    Emplacement/@emplacementID"/></xsl:attribute>
90          <xsl:attribute name="profileIDRef"><xsl:value-of select="$var.profile
    "/></xsl:attribute>
91          <!-- Processing all profile definitions-->
92          <!--Copy the Shared resources-->
93          <SharedResources>
94            <xsl:apply-templates select="//me:Emplacement/me:SharedResources"/>
95            <!--TODO Pick out the selective properties defined by the Profiles
    -->
96          </SharedResources>
97          <!--Copy the Shared resources-->
98          <InterfaceStandardsInfo>
99            <xsl:copy-of select="//me:Emplacement/me:InterfacePorts"/>
100           <xsl:copy-of select="//me:Emplacement/me:Interfaces"/>
101           <xsl:copy-of select="//me:Emplacement/me:StdBodies"/>
102         </InterfaceStandardsInfo>
103         <Control>
104           <Skills>
105             <!--Picks the wanted profile-->
106             <xsl:apply-templates select="key('keyProfile', $var.profile)"
    mode="forSkills">
107               <xsl:sort select="@id" order="ascending"/>
108             </xsl:apply-templates>
109           </Skills>
110           <Implementation/>
111         </Control>
112         <Mechanic>
113           <Interfaces>
114             <!--Picks the wanted profile-->
115             <xsl:apply-templates select="key('keyProfile', $var.profile)"
    mode="withIFType">
116               <xsl:with-param name="ifType">MECHANICAL_PORT</xsl:with-param>
117               <xsl:sort select="@interfacePortRef" order="ascending"/>
118               <xsl:sort select="@interfacePortRefQualifier" order="ascending"
    />
119             </xsl:apply-templates>
120           </Interfaces>
121         </Mechanic>
122         <Service>
123           <Interfaces>
124             <!--Picks the wanted profile-->
125             <xsl:apply-templates select="key('keyProfile', $var.profile)"
    mode="withIFType">
126               <xsl:with-param name="ifType">SERVICE_PORT</xsl:with-param>
127               <xsl:sort select="@interfacePortRef" order="ascending"/>
128               <xsl:sort select="@interfacePortRefQualifier" order="ascending"
    />
129             </xsl:apply-templates>
130           </Interfaces>
131         </Service>
```

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3779-0
ISSN 1459-2045